Advanced Systems Lab Report

Autumn Semester 2018

Name: Nicolas Küchler
Legi: 14-712-129

Grading

Section	**Points**
1	
2	
3	
4	
5	
6	
7	
Total	

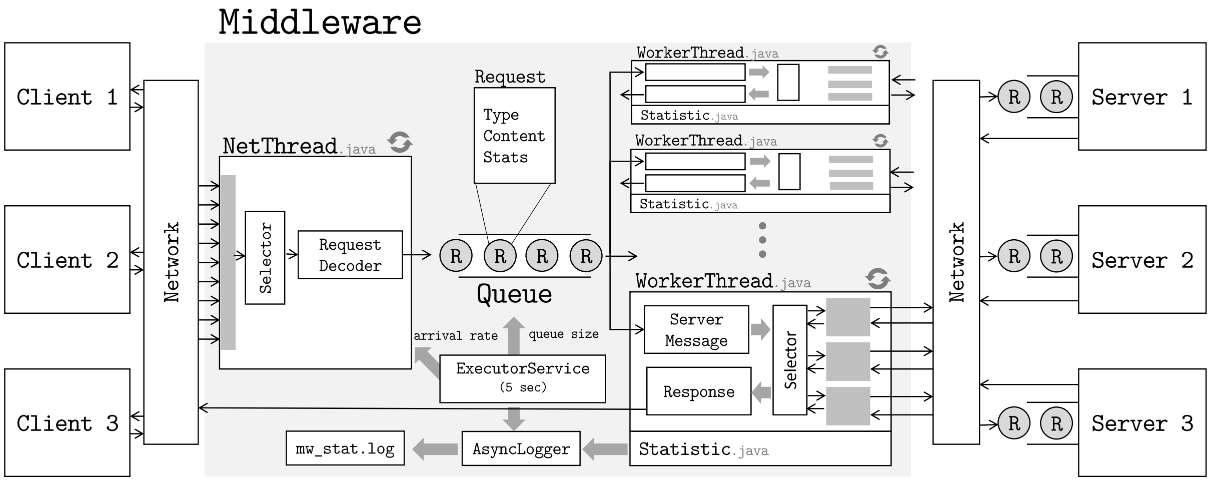


Figure 1: System Overview

1 System Overview (75 pts)

The system is a middleware (MW) platform for the popular main-memory key-value store *memcached* [2]. It supports up to 3 *memcached* servers and allows to balance the read workload by replicating all writes to all servers.

1.1 Middleware Architecture

1.1.1 High-Level Overview

The middleware system is composed of three main components:

- A single net-thread accepting and decoding requests from clients on a TCP port and putting them into an internal request queue.
- A request queue buffering requests for the processing by a worker-thread.
- Worker-threads processing the requests from the queue according to their type.

Start The middleware can be configured at startup time using command line arguments. When the middleware is launched, the configured number of worker-threads (-t) are spawned and each of them builds a dedicated TCP connection to each *memcached* server (-m). The connections are kept open until the middleware is stopped. At startup time the net-thread starts to listen on the specified TCP port (-l -p) for client requests. Additionally there are 2 modes (-s) for handling multi-GET requests, which are GET requests with multiple keys.

Run The net-thread *(NetThread.java)* listens on a configured TCP port for client requests. The different client channels are handled using a java nio *Selector*. After decoding the incoming request in the net-thread (section 1.1.3) they are put into a queue *(LinkedBlockingQueue.java)*. The worker-threads *(WorkerThread.java)* take requests from the queue and process them according to their their type. (section 1.1.4). When the processing of a request is completed, a response is sent to the client using the channel which was opened initially in the net-thread.

Shutdown In order to support a graceful shutdown of the middleware, a shutdown hook is registered at startup time. As soon as the hook caught a Linux kill command, it interrupts the net-thread and all worker-threads and ensures all logs have been written to the respective log file. Afterwards the middleware is stopped.

1.1.2 Request

The middleware only supports SET, GET and multi-GET operations according to the protocol specification in [3]. Logically each request from a client is represented using an instance of *SetRequest.java*, *GetRequest.java* or *MultiGetRequest.java*. They all inherit basic functionality from *AbstractRequest.java*. These request objects are created in the net-thread by the decoder (Section 1.1.3) and placed into the queue where they wait to be processed by a worker-thread (Section 1.1.4). Apart from containing the operation command, the request object also serves as a container for the TCP socket channel to the client and for timestamps collected at different points in the system (Section 1.2.1). This allows to measure how much time the request spent in each part of the system.

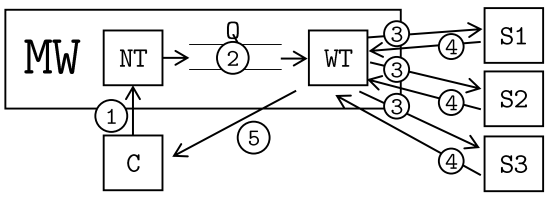
1.1.3 Request Decoding

The net-thread *(NetThread.java)* extends the java *Thread* class. In the *run()* method the net-thread is using an nio *Selector* to handle multiple client channels in a single thread. The net-thread iterates over all channels that are ready and loads the content into a buffer *(ByteBuffer.java)*. From there the incoming requests are decoded using the function *decode(ByteBuffer buffer)* of the request decoder *(RequestDecoder.java)*. Since every request needs to pass this function, the decoding is done on byte level to avoid unnecessary overhead. Apart from checking that the client request is supported by the middleware, the decoder also verifies that the request is complete. In case the decoder encounters an unsupported operation or a malformed request, the request is discarded until a newline character is read. Then an error is returned to the client and a log entry is created. If a request is not complete yet, the decoder signals the net-thread that more is expected and the net-thread will append the next content arriving from the client channel to the same buffer to complete the request. In the following the decoding of the three different request types is explained in detail.

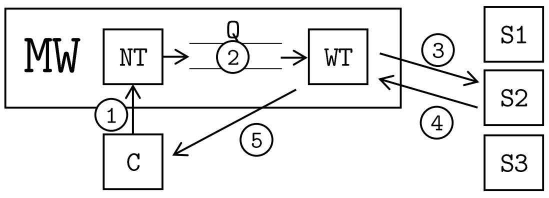
SET cmd: `set <key> <flags> <exptime> <bytes>[noreply]\r\n<datablock>\r\n`
After reading the "set" in the beginning of the command, the decoder skips over key, flags and expiration time before reading the bytes field to determine whether the data block is complete. This is done by checking that the size of the buffer content matches the expected size and ends with the delimiter sequence `\r\n`. If the request is complete, a *SetRequest.java* is created.

GET / MULTI-GET cmd: `get <key>\r\n` or `get <key1> <key2> ... <key10>\r\n`
After reading the "get" in the beginning of the command, the decoder determines the number of keys by counting the number of whitespaces and checks that the request is complete by checking that it ends with `\r\n`. If there is only a single key, then a *GetRequest.java* is created, otherwise a *MultiGetRequest.java* is created.

For GET and multi-GET requests the decoder assigns each request a server (or for sharded multi-GET multiple servers) in a round-robin scheme. (more on workload balancing in section 1.1.5)



(a) SET and sharded multi-GET



(b) GET and non-sharded multi-GET

Figure 2: Schematic overview of different request types between client (C), net-thread (NT), queue (Q), worker-thread (WT) and server (S).

1.1.4 Request Processing

The worker-thread (WorkerThread.java) extends tha java *Thread* class and takes a request from the queue with a blocking operation that waits until a request is available. Afterwards the worker-thread calls the function *getServerMessages()* of the request object. This method is responsible for assembling all server messages (*ServerMessage.java*) according to the request type. The worker-thread sends these messages to the specified servers and afterwards waits for all servers to respond. The different channels to the servers are monitored using an nio *Selector*. Whenever a response of a server arrives in the worker-thread, it is given to the request object using the function *putServerResponse(serverId, buffer)*. The request object collects all responses and as soon as all arrived, a response for the client is assembled and transmitted through the open socket channel. After this general overview of the request processing, in the following the processing of the three different request types is explained in detail.

SET The SET request command obtained from the client is forwarded to all servers. If all servers answer with `STORED\r\n`, then the confirmation is forwarded to the client. If one or multiple servers answer with an error message, the error message which arrived last is relayed back to the client. In this case a warning is written to a log file. (Fig. 2a)

GET The GET request command received from the client is relayed to the server determined by the round-robin scheme. The response of the server is directly forwarded to the client. (Fig. 2b)

MULTI-GET The processing of a multi-GET request is different depending on the sharded/non-sharded mode which can be configured at the startup time. In case of non-sharded mode, the processing is identical to a GET request where the message is sent to one server and the response is directly relayed to the client (Fig. 2b). In case of sharded mode, the request is split evenly into three smaller requests such that the difference in the number of keys per server is maximally one (i.e. for 7 keys, 2 servers receive 2 keys each and 1 server receives 3 keys). Each part is sent to the server according to the round-robin scheme. Whenever a server response arrives in the worker-thread, it is fed back to the multi-GET request object where it is stored until all servers responded. Then the client response is assembled by splitting and reordering the individual responses of the servers before sending it to the client (Fig. 2a).

1.1.5 Workload Balancing

The round-robin scheme applied for GET and multi-GET requests allows to balance the read workload among multiple servers.

As described in 1.1.3 each GET and multi-GET request receives one or multiple server ids from the decoder in the net-thread. For GET and multi-GET requests in non-sharded mode this is only one server id. For multi-GET requests in sharded mode this is a set of server ids. (if the number of keys is larger than the number of servers, then this will be all servers).

Since the net-thread is a singleton and the request queue offers first come first served processing of requests, the order in which the requests are processed does not change. Hence when neglecting network related effects, it can be expected that the work will be distributed evenly among the available servers.

Another form of work balancing takes place with the number of worker-threads. For a single worker-thread there is no balancing. But as the number of workers increases, also the concurrency in the system increases because multiple requests can be processed in parallel.

The balancing of the read workload comes at the cost of the write workload. Since the value in each SET request is replicated to all *memcached* servers, the total server service time is determined by the last server responding.

1.1.6 Logging Infrastructure and Statistics

Logging in the system uses *log4j2* asynchronous loggers [1]. The middleware produces two log files: *mw_out.log* containing all info, warning and error messages and *mw_stat.log* including all performance statistics of the middleware.

Each worker-thread contains a separate statistic unit *(Statistic.java)* that collects statistics for requests processed within this thread. After the processing of each request the worker-thread calls the method *update(request)* to incorporate the new request into the statistics.

The statistic unit is collecting online averages and sample standard deviations of all metrics (1.2.1) over a five second window using *Welford's Algorithm.* [6] The algorithm provides a method to calculate the average $\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$ and the variance $s_n^2 = \frac{M_{2,n}}{n-1}$ where $M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)$ in an online and numerically stable fashion. Every 5 seconds $\bar{x}_n$, $M_{2,n}$ and n of every metric are logged to *mw_stat.log* and then reset to 0 afterwards. These measurements can then be aggregated off-line into 3 error metrics. First the standard deviation within every 5 second window which gives insight into how stable the metric is in the 5 second window. Secondly the standard deviation over the 5 second averages where a warmup and cooldown phase are filtered out. This allows to check if the execution over the 60 seconds was stable. Third the standard deviation over the average of each repetition which gives the possibility to check how much a measurement varies when repeating an experiment.

The statistic unit of each worker-thread also keeps track of a histogram of the response time distribution in $100\mu s$ steps. This histogram is flushed to *mw_stat.log* and reset every 5 seconds. The individual histograms of each worker-thread and time window are then aggregated off-line which also allows to filter out measurements from the warmup and cooldown phase.

In addition there is a separate executor service *(ScheduledExecutorService.java)* in the MW that executes every 5 seconds a runnable collecting statistics about the arrival rate in the net-thread and the internal request queue.

1.2 Experimental Setup

The behaviour of the system under different configurations and user loads was analysed in a simulation on the *Azure* cloud. The measurements were obtained using up to 8 VMs:

- 3 Linux Client VMs of type Basic A2 (2 vcpus, 3.5 GB memory) each running one or two instances of the *memtier benchmark* (version 1.2.15)
- 2 Linux Middleware VMs of type Basic A4 (8 vcpus, 14 GB memory) running the MW
- 3 Linux Server VMs of type Basic A1 (1 vcpus, 1.75 GB memory) each running *memcached* (version 1.4.25) with 1 thread

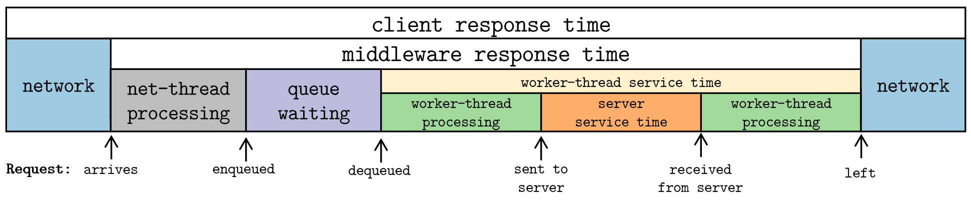


Figure 3: components of client response time

Memtier benchmark is a command line utility for load generation and benchmarking key-value stores [4]. When using multiple *memtier* instances, the number of clients in the system is defined as the number of memtier instances times the number of threads per instance times the number of virtual clients per thread. Throughout all experiments a constant value size of 4096B was used and the number of keys was limited to 10000. All SETs have a long expiry time, such that keys are not evicted during the experiment.

1.2.1 Metrics

The performance metrics listed in the table below provide an overview over the collected statistics to evaluate the performance of the system. Figure 3 shows how they are related. Throughout the report, the origin of the data (i.e. MW or client) is indicated with a label to avoid ambiguity.

Origin	Metric	Description
client	- avg throughput - avg response time - response time histogram	as measured by *memtier benchmark* as measured by *memtier benchmark* as measured by *memtier benchmark*
mw	- avg throughput - avg response time - avg net-thread processing time - avg queue waiting time - avg worker-thread service time - avg worker-thread processing time - avg server service time - avg queue length - avg request arrival rate - response time histogram	per 5 second window time between request arrived in MW and left MW time between request arrived and was enqueued time request was in the queue time between dequeuing and leaving MW "worker-thread service time" - "server service time" time *memcached* needed to process the request (measured for each server individually and in total) sampled every 5 seconds in net-thread per 5 second window in 100 μs steps
network	- round trip time - bandwidth	delay between different VMs network capacity between different VMs

1.2.2 Simulation

All experiments were orchestrated with a *python* script outlined in algorithm 1 running on a local machine. Temporary *ssh* connections were used to start, initialize and stop the run of an experiment. Before every set of experiments the network topology of all involved VMs was analysed by running a bandwidth test using *iperf*. This in combination with the 4096B value size results in an estimate for the maximal achievable throughput for the given VM configuration and helps to identify when the network bandwidth is the bottleneck of the system. For every repetition of an experiment the *memcached* server and the MW software were restarted. After restarting the *memcached* server a *python* script populated the key-value store with every possible key in order to prevent an influence of cache misses on the performance. After each run the resulting log files were transferred to the local machine and stored in a *MongoDB*. The obtained results are filtered (remove 10 second warmup and cooldown phase) and later aggregated off-line using *MongoDB* queries. All visualizations were done using *matlibplot*.

After repeating an experiment 3 times, the results were validated by checking that the coefficient of variation of the throughput over the 60 second runtime and the coefficient of variation over the repetitions did not exceed a threshold. If necessary additional repetitions were scheduled to account for external factors on the cloud impacting the performance.

```
foreach set of experiments do
    - measure bandwidth and ping of the current network topology using iperf and ping
    foreach experiment do
        foreach repetition do
            - start all memcached instances with one thread
            - initialize all memcached instances by populating the key value store to avoid cache misses
            - start all middlewares according to the experiment configuration
            - start all client benchmarks for 80 seconds with a value size of 4096B according to the config
            - wait 80 seconds
            - stop all middlewares
            - stop all memcached instances
            - transfer results from VMs to the local machine using SCP
            - parse log files and store results in local MongoDB
        end
        - validate results and possibly run additional repetitions
    end
end
```

Algorithm 1: Each section of the report represents a set of experiments where different configurations were evaluated using at least 3 repetitions with a stable runtime of 1 minute each.

Apart from the set of experiments for the baseline without middleware in section 2 and GET/multi-GET in section 5, all experiments were run on the same set of VMs without restarting them. This allows a comparison of results not only within a set of experiments but also between different sections.

2 Baseline without Middleware (75 pts)

This section establishes a performance baseline of the *memtier benchmark* clients and *memcached* servers in the cloud.

2.1 One Server

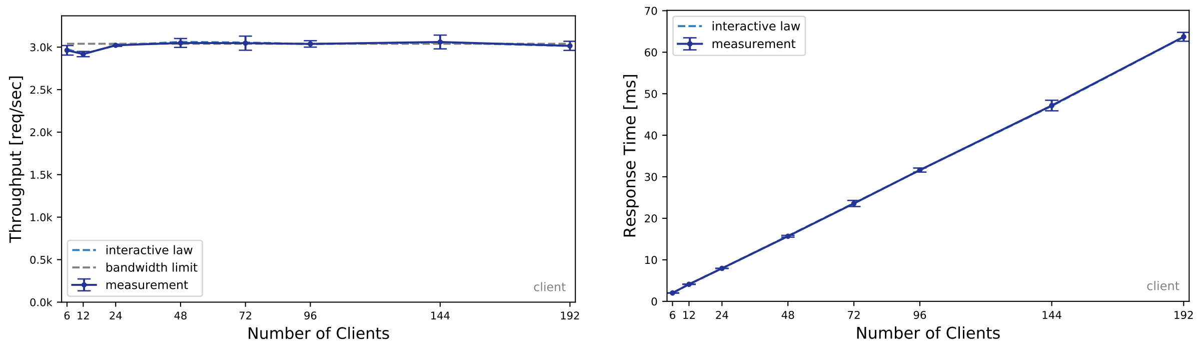
In this set of experiments the behaviour of the system with 3 load generating VMs and 1 server VM is analysed. The detailed experiment configuration is presented in the table below.

3 client VMs and 1 server VM

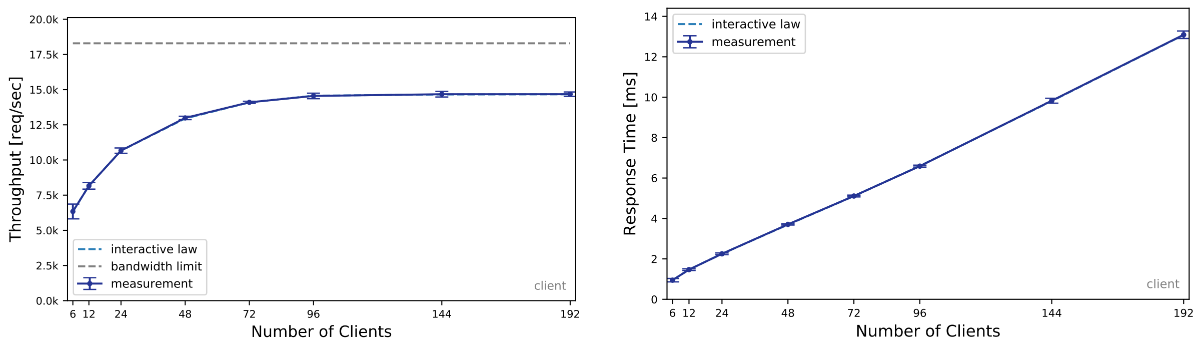
Instances of memtier per machine	1
Threads per memtier instance	2
Virtual clients per thread	[1, 2, 4, 8, 12, 16, 24, 32]
Workload	Write-only and Read-only

Apart from the client measurements for throughput and response time, the graphs also contain the throughput and response time derived from the interactive law[1] with a client thinking time of $Z = 0$ and a maximal achievable throughput due to the network bandwidth limit between the VMs when using values of size 4096B.

[1]the interactive law $R = \frac{N}{X} - Z$ shows the relationship between response time R and throughput X in a closed system with N clients and a client thinking time of Z



(a) read-only workload



(b) write-only workload

Figure 4: Throughput and response time measurements of the system with one server verified with the interactive law as a function of the number of clients. The error metric is the sample standard deviation over the repetitions and the maximal throughput due to the network bandwidth is shown.

2.1.1 Explanation

Read-Only Workload As shown in figure 4a, the throughput saturation for the read-only workload is already reached at 6 clients because the upload bandwidth of the single server VM limits the throughput. The server VM upload bandwidth was measured to be 99.6 MBit/sec. When using a value size of 4096B, this results in a maximum possible throughput of 3040 ops/sec for a read-only workload because all values need to be sent from the server VM to one of the 3 client VMs. Consequently increasing the number of clients has almost no effect on the throughput while the response time grows linearly because more clients result in more requests on the server that need to wait to be sent through the bottleneck to a client VM.

Write-Only Workload In figure 4b it can be observed that for a write-only workload the throughput saturation is reached at 72 clients. In this experiment the bottleneck is not the bandwidth but rather the *memcached* server. As expected the throughput first increases in the number of clients while the *memcached* servers are still under-saturated. The response time of the *memcached* server does not show the typical behaviour between the under-saturated and saturated phase, because even though the throughput still increases drastically, the response time is already showing a linear increase without a clear knee. However, even though using 96 clients would result in a marginally higher throughput, the increase in response time does not justify a user load of more than 72 clients as the maximum throughput and hence with more than 72 clients the system is saturated. The over-saturated phase with decreasing throughput is not shown because *memcached* is able to keep the throughput stable even for a large user-load.

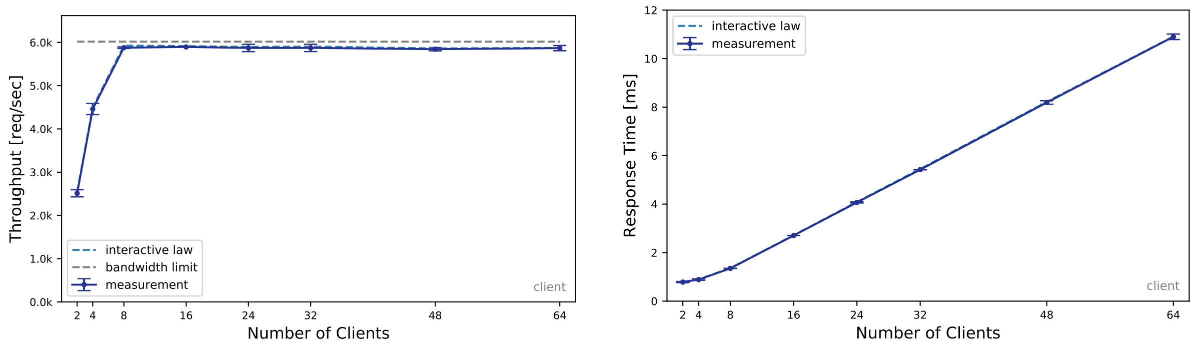
2.2 Two Servers

This set of experiments analyses the behaviour of the system with 1 load generating VM and two server VMs. The detailed experiment configuration is presented in the table below.

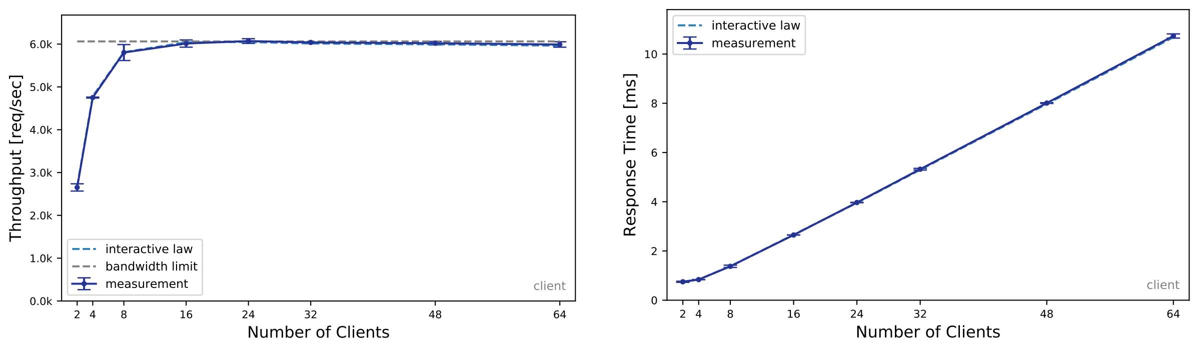
1 client VM and 2 server VMs

Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	[1, 2, 4, 8, 12, 16, 24, 32]
Workload	Write-only and Read-only

As in the previous section, the throughput and response time derived from the interactive law with a client thinking time of $Z = 0$ and the network bandwidth limit for the throughput presented next to the measurements obtained from the client.



(a) read-only workload



(b) write-only workload

Figure 5: Throughput and response time measurements of the system with one load generating VM verified with the interactive law as a function of the number of clients. The error metric is the sample standard deviation over the repetitions and the maximal throughput due to the network is shown.

2.2.1 Explanation

Read-Only As with a single server VM, the throughput of the read-only workload with 2 servers is also bound by the total upload bandwidth of the server VMs. This becomes evident when considering that two server VMs have a measured total upload bandwidth capacity of 199 MBit/sec which results in an estimate of 6062 ops/sec for the maximally achievable throughput when using values of 4096B.

The optimal number of clients for a read-only workload would be somewhere in between 4 and 8 clients because when using 8 clients the bandwidth bottleneck is already affecting the throughput and response time. However, from the evaluated number of clients it cannot be justified to state that 4 clients is the load with maximum throughput because the increase in throughput of around 1500 ops/sec with 8 clients simply outweights the cost of only 0.45 ms more in response time on average. Consequently the point of throughput saturation is reached at 8 clients. For fewer number of clients the system is under-saturated and for more it is saturated. The over-saturation phase with decreasing throughput is omitted. (Fig. 5a)

Write-Only Using a write-only workload with values of 4096B the situation is similar but here instead of the upload bandwidth of the server VMs, the bottleneck is the upload bandwidth of the load generating client VM. The measured 199 MBit/sec total upload bandwidth limits the throughput of the system to 6062 ops/sec. So as a consequence the throughput saturation is reached at 8 clients as shown in figure 5b. The increase in response time between 4 and 8 clients is still justified for the additional throughput obtained with 8 clients. But the additional 200 ops/sec for 16 clients is not enough to outweigh a response time that is almost twice as long.

2.3 Summary

Maximum throughput of different VMs.

	Read-only workload	Write-only workload	Configuration gives max. throughput
One memcached server	2962 ops/sec	14097 ops/sec	6 clients for read-only 72 clients for write-only
One load generating VM	5878 ops/sec	5802 ops/sec	8 clients for read-only 8 clients for write-only

Comparison of one memcached server vs. one load generating VM For a read-only workload both experiments are bound by the network upload bandwidth of the server VM(s). Despite the fact that they are both network bound, they have a different maximum throughput because having an additional server in the second set of experiments gives two times the upload bandwidth between server and client VMs and thus resulting in approximately two times the throughput.

The different throughput for write-only workloads is explained by considering that only the second experiment is bound by the network. Namely by the upload bandwidth of the client VM. The bottleneck in the first experiment is *memcached* which saturates for around 72 clients.

Comparison of workloads In the first set of experiments with one *memcached* server the throughput is different between write and read workload. This is primarily because only the read workload is network bound. For a SET request the server only needs to transmit the 8B confirmation message `STORED\r\n` to the client compared to a GET request where the server needs to transport a message with at least 4096B.

Despite the fact that in the second set of experiments with only one load generating VM the throughput for read-only and write-only is almost identical, the bottleneck is a different part of the system. For read-only it is the upload bandwidth of the two server VMs and for the write-only workload it is the upload bandwidth of the load generating client VM.

Key take-away messages

- a single server VM cannot handle more than 3000 ops/sec in a read-only workload
- a single client VM cannot simulate a higher write-only workload than 6000 ops/sec
- in a write-only workload the point of saturation for 1 server VM is around 14000 ops/sec

3 Baseline with Middleware (90 pts)

In this set of experiments the impact of the number of clients on the performance of the system is studied when using 3 load generating VMs and 1 *memcached* server.

All measurements are validated using the interactive law, which describes the relationship between throughput X and response time R in a closed system with N clients where each client has a client thinking time of Z (Eq. 1). For all experiments a client thinking time of $Z = 0$ is used. The interactive law is expected to hold for all experiments.

$$R = \frac{N}{X} - Z \qquad\qquad Error = R_{client} - (\frac{N}{X_{mw}} - Z) \tag{1}$$

However, the interactive law cannot be applied directly to validate the MW throughput and response time measurements. This is due to the fact that the middleware cannot measure the time a request needs for the transport over the network between client and MW. This results in a shorter response time measurement in the middleware than the actual response time measured on the client. Hence when using the middleware response time, this would result in a throughput that is higher than the measured throughput. To circumvent this issue while still being able to use the interactive law as a sanity check for the collected data, the response time as measured on the client is used instead. In addition a manual check was performed to verify that the response time differences between client and middleware measurements is approximately the round trip time between client VM and middleware VM: $R_{client} - R_{mw} \approx rtt$.

3.1 One Middleware

Three load generating VMs are connected to one middleware handling requests for a single server. The number of clients is varied between 6 and 288 depending on the saturation of the system with 8, 16, 32 and 64 worker threads inside the middleware for both a read-only and write-only workload. The details of the configuration are shown in the table below. The interactive law was checked and holds for all experiments in this section (Tab. A.0.1)

3 client VMs, 1 middleware VM and 1 server VM

Instances of memtier per machine	1
Threads per memtier instance	2
Virtual clients per thread	[1, 2, 4, 8, 12, 16, 24, 32, 48]
Workload	Write-only and Read-only
Worker threads per middleware	[8, 16, 32, 64]

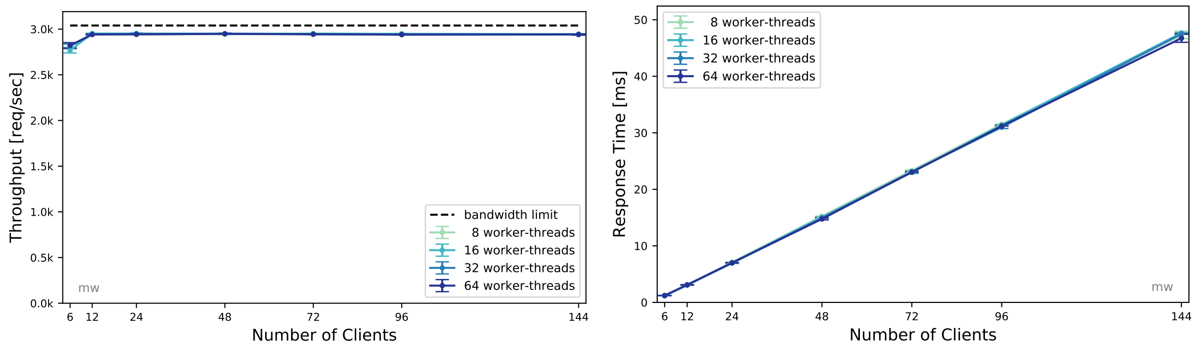


Figure 6: Throughput and response time in read-only workload for the system with 1 MW as a function of number of clients. The sample standard deviation over the repetitions is used as the error metric to show that the measurements are stable.

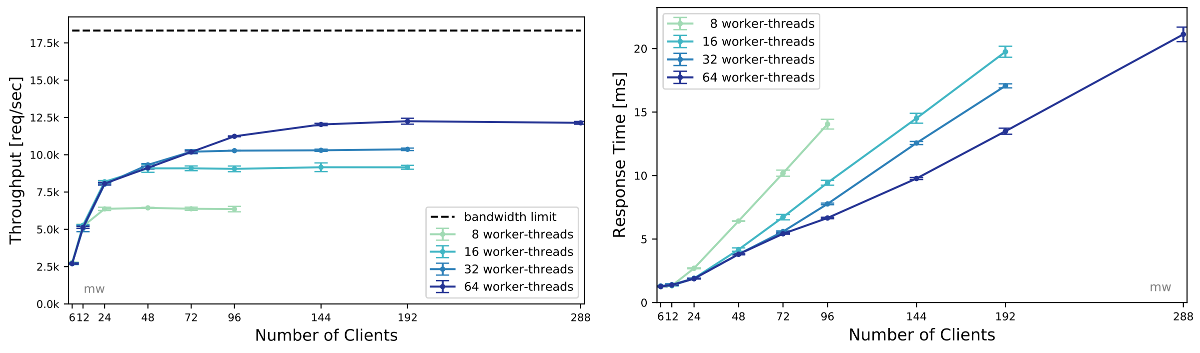


Figure 7: Throughput and response time in write-only workload for the system with 1 MW as a function of number of clients. The sample standard deviation over the repetitions is used as the error metric to show that the measurements are stable.

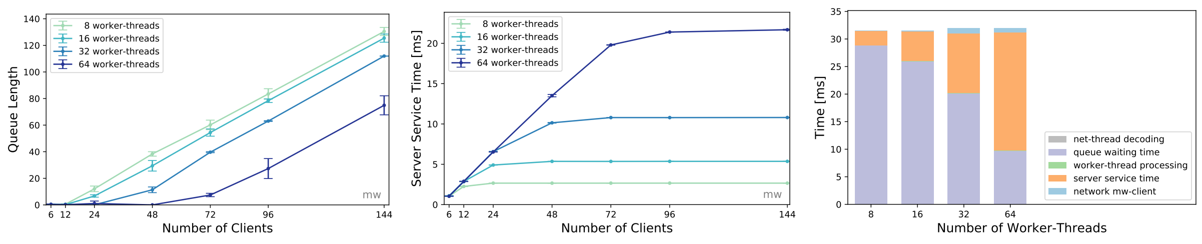
3.1.1 Explanation

Read-Only Workload As described in section 2, the read-only workload with a single *memcached* server VM and values of size 4096B is bound to around 3000 ops/sec by the upload bandwidth of the server VM. This phenomena is also clearly visible in figure 6 when using a single middleware in between 3 client VMs and a server VM. Consequently the point of throughput saturation of a read-only workload is reached already with 6 clients, independent of the number of worker-threads. This also manifests itself in the response time that increases linearly in the number of clients due to the extended waiting period on the server VM.

In the following it is analysed how this network bandwidth bottleneck manifests itself in the middleware measurements. As long as the number of clients is less than the number of workers per MW, the request queue in the middleware is empty (Fig. 8a). This is because the number of clients in a closed system limit the number of requests that can be in the MW at the same time. So when there are more worker-threads than possible requests in the system, every request placed in the queue is processed immediately by one of the idle workers. This is a fact that holds independently of the network bottleneck.

Figure 6 shows that the response time is not affected by the number of worker-threads because already using 8 workers is enough in combination with 12 clients to reach the network bottleneck. So adding more workers cannot improve the throughput. However, the response time increases in the number of clients because the number of requests per second that can be transported from the server VM to the MW remains fixed. Consequently with a greater user-load, requests need to wait longer in some part of the system before the bottleneck, leading to a longer response time. This also manifests itself in the different components of the response time. The queue waiting time and the server service time are the two dominant factors of the response time as measured on the client while the network-, the net-thread decoding- and worker-thread processing time are negligibly small (Fig. 8c). As the number of worker increases, requests spend less time in the queue and instead stay longer on the server VM where they are waiting in front of the network bottleneck. So the number of workers does not influence the response time but it influences if the requests are queued in the middleware or on the server (Fig. 8c).

The server service time increases up to the point where are all workers are busy and the queue starts to fill up. Afterwards it remains constant because the number of workers in the middleware limit the number of requests that can be concurrently at the server VM and hence need to be sent through the network bottleneck. (Fig. 8b)



(a) **(b)** **(c)**

Figure 8: Length of queue and server service time as a function of number of clients with sample standard deviation over the repetitions as the error metric and client response time components for 96 clients in a read-only workload with 1 MW.

Write-Only Workload The data in figure 7 indicates that the throughput saturates for a different number of clients when varying the number of worker-threads. This is because there is a trade-off between server service time and queueing time in the MW controlled by the number of workers. For fewer workers, requests start to queue up already with a smaller number of clients because the number of worker-threads controls how many requests can be sent to the server concurrently. However with more requests at the server concurrently, the server also has a longer server service time for each of them. The throughput saturation for 8 workers is reached at 24 clients, for 16 workers at 48 clients, for 32 workers at 72 clients and for 64 workers at 144 clients.

As the user-load is increased, eventually the waiting time in the queue becomes the dominant factor in the response time for every number of worker-threads. This is because when all workers are busy waiting for a response from the server, each new request remains longer in the queue. This becomes evident in the decomposition of the response time in figure 11. The component utilization figure 10 shows that the bottleneck in the system is the number of worker-threads that are waiting concurrently for a response of the server. This suggests that increasing the number of worker-threads would further benefit the throughput. However, the increasing server service time for an individual request with a larger number of worker-threads poses a limit on this throughput increase. When there are more clients than worker-threads, the number of requests in the queue is approximately equal to the difference between number of clients and number of worker-threads. As in the read-only workload the net-thread decoding and worker-thread processing time remain negligible factors in the response time.

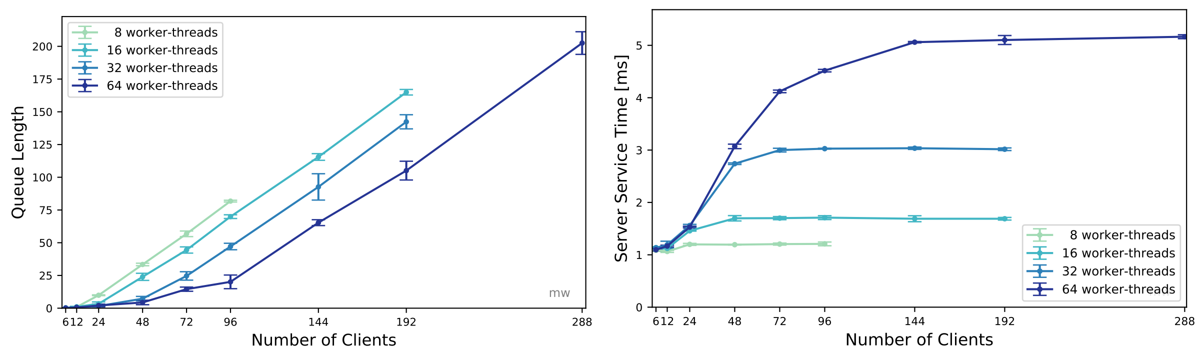
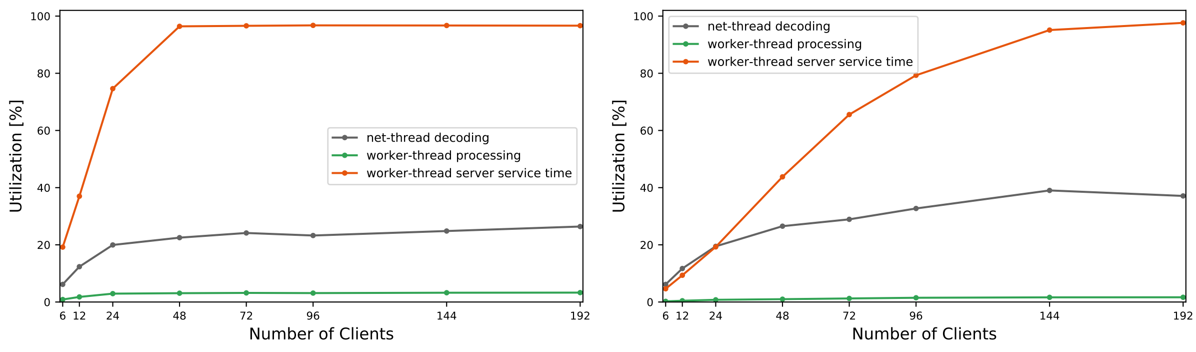
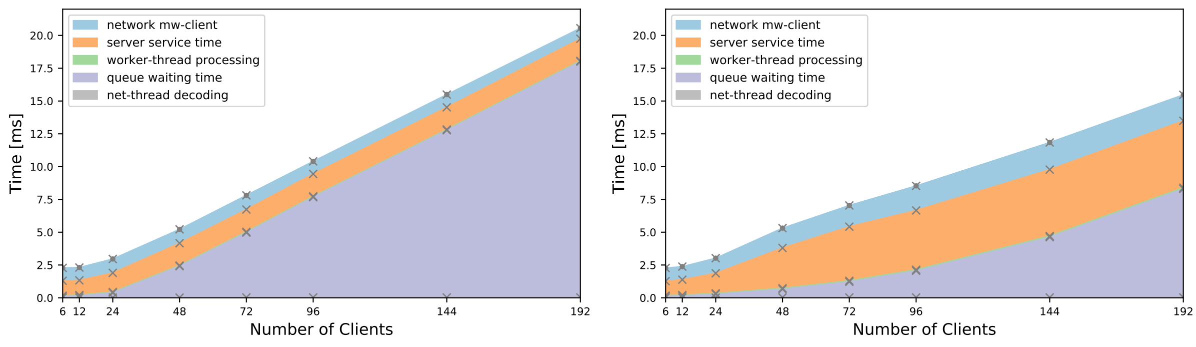


Figure 9: Queue length and average server service time per request as a function of number of clients for a write-only workload in a system with 1 MW. Both graphs show the sample standard deviation over the repetitions as error metric.



(a) 16 worker-threads **(b)** 64 worker-threads

Figure 10: Utilization $\frac{\text{busy time}}{\text{total time}}$ as a function of number of clients in a write-only workload with 1 MW.



(a) 16 worker-threads **(b)** 64 worker-threads

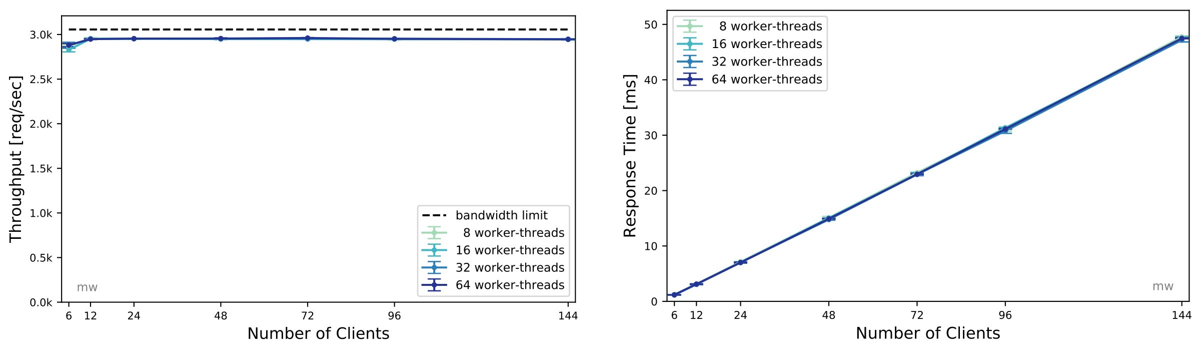
Figure 11: Response time components as a function of number of clients for SET requests with 1 MW.

3.2 Two Middlewares

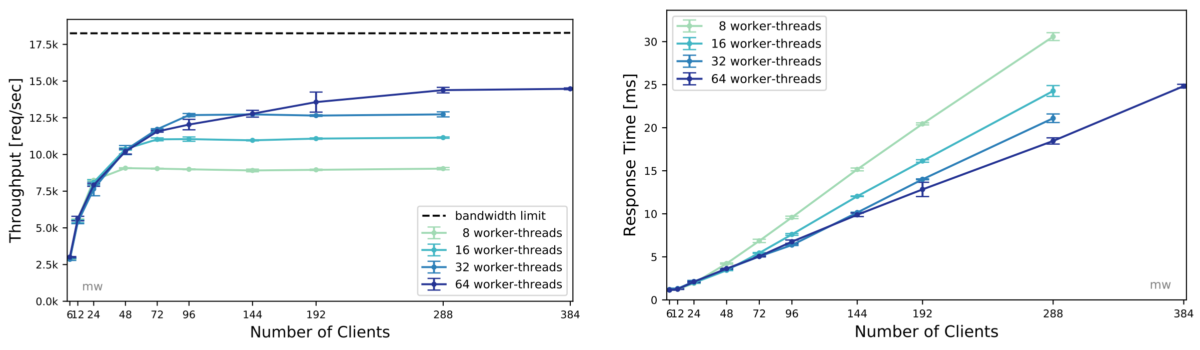
In this set of experiments, 3 load generating VMs are connected to 2 MWs handling requests for a single server. The number of clients is varied between 6 and 384 depending on the saturation of the system with 8, 16, 32 and 64 worker-threads per MW for both a read-only and write-only workload. The details of the configuration are shown in the table below. The interactive law was checked and holds for all experiments in this section (Tab. A.0.2)

3 client VMs, 2 middleware VMs and 1 server VM

Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	[1, 2, 4, 8, 12, 16, 24, 32, 48, 64]
Workload	Write-only and Read-only
Worker threads per middleware	[8, 16, 32, 64]



(a) read-only workload



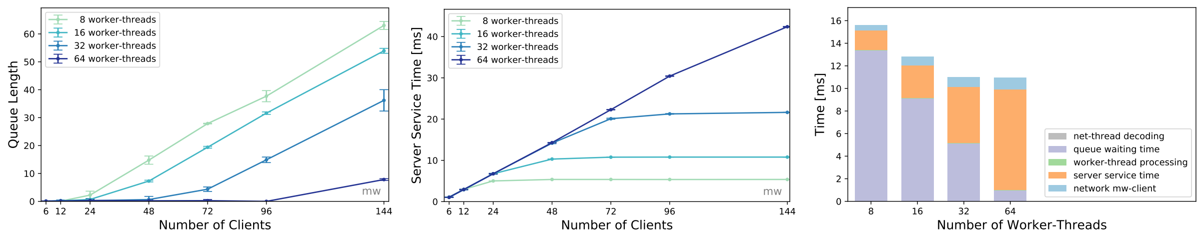
(b) write-only workload

Figure 12: Throughput and response time for the system with 2 MWs as a function of number of clients. The sample standard deviation over the experiment repetitions is used as the error metric.

3.2.1 Explanation

Read-Only Workload As in the setting with 1 MW, the read-only workload is bound by the upload bandwidth of the server. Since the bottleneck is not related to the MW, it is no surprise that adding an additional MW does not change the observed behaviour and the throughput saturation is still reached at 6 clients independent of the number of workers (Fig. 12a). The response time also shows the same behaviour as in the case with 1 MW for the identical reasons.

However considering the internal MW measurements, there are a few differences. Since the two MWs share the workload, each MW has only half of the clients. Thus requests are only starting to be queued up in the request queue when there are two times more clients than workers per MW (Fig 13a). The same holds for the server service times that start becoming constant in the number of clients when there are more than two times more clients than worker-threads due to the same reasons as outlined in section 3.1. The trade-off between queue waiting time and server service time as the dominating components of the response time is also evident in the setting with 2 MWs (Fig. 13c) and so the number of workers still does not influence the total response time but it influences in which part of the system the requests are queued.



(a) (b) (c)

Figure 13: Queue length and server service timeas a function of the number of clients and client response time component analysis for 144 clients in a read-only workload with 2 MWs.

Write-Only Workload In the write-only workload the system with 2 MWs behaves in a similar way as the system with 1 MW. As shown in figure 15 the bottleneck remains the number of worker-threads that are waiting for a server response. However, the baseline without middleware suggests that for 72 clients the throughput saturation is reached. This means that using more than 72 workers in total does not lead to a higher throughput. The server service time still depends only on the number of busy worker-threads (Fig. 16). However, by adding an additional MW the total number worker-threads in the system is multiplied by two and hence the maximal throughput achieved is higher for the same number of worker-threads per MW. For 8 worker-threads per middleware the point of saturation is reached with 48 clients, for 16 workers with 72 clients, for 32 workers with 96 clients and for 64 workers with 192 clients.

The collected data indicates that the total number of worker-threads in the system is the decisive factor for performance up to the point where the total number of workers reaches the saturation of the server. So essentially the system with 2 MWs has a similar performance to the system with 1 MW with two times as many worker-threads. (i.e. system with two middlewares with 32 workers per MW has approximately the same performance as the system with 1 MW and 64 worker-threads) This results from the fact that neither the net-thread nor the middleware VM network bandwidth is the bottleneck and so by duplicating the middleware VM the only performance gain is the consequence of more worker-threads in the system.

The total number of worker-threads in the system affect the performance because the server service time depends only on the number of worker-threads that are sending requests concurrently (Fig. 9 and 14). Consequently each request needs to wait in the queue for approximately the same time and hence the total response time is the same leading to the same throughput in systems with the identical number of total worker-threads (Fig. 7 and 12b). The system with 1 MW has on average two times as many requests in the queue compared to each queue in the system with 2 MWs but the total number of requests in the whole system that are waiting in a middleware queue is the same when the total number of workers is identical (Fig. 9 and 14).

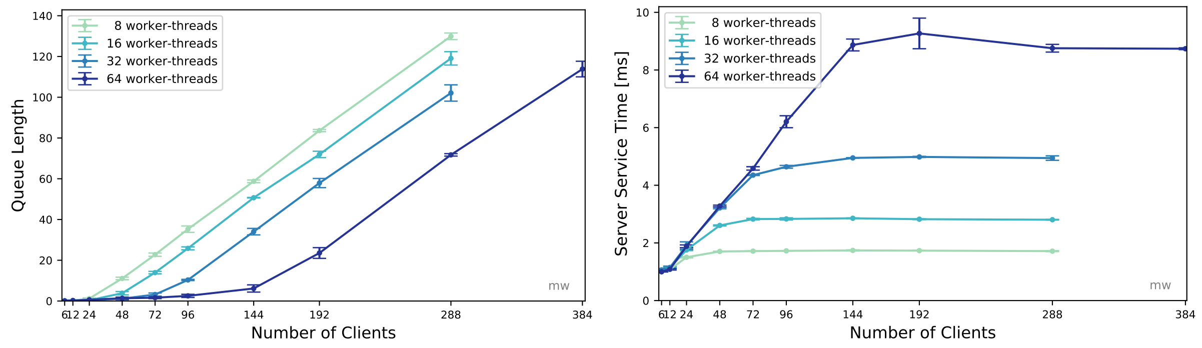
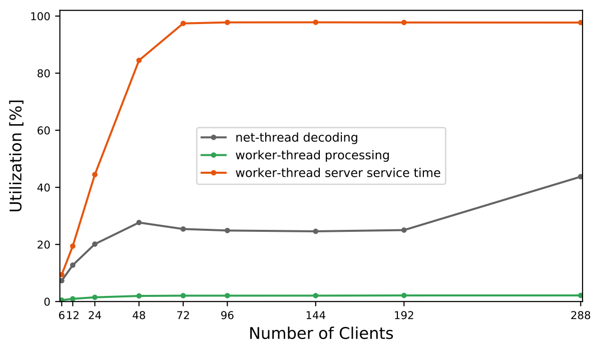
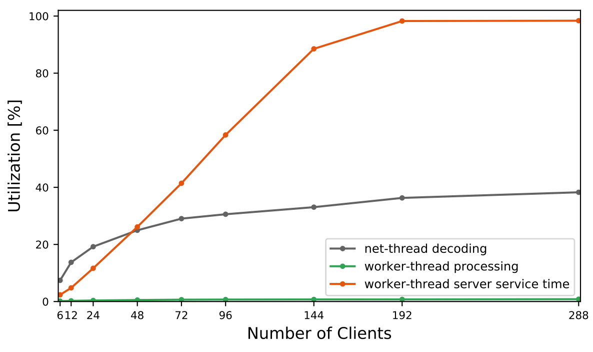


Figure 14: Queue length per MW and server service time per request in a write-only workload with 2 MWs. Boths graph show the sample standard deviation over the repetitions as error metric.

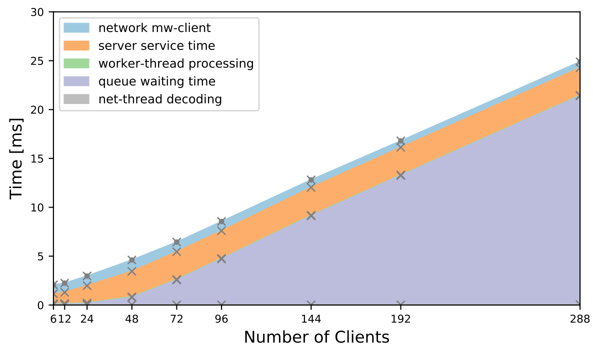


(a) 16 worker-threads

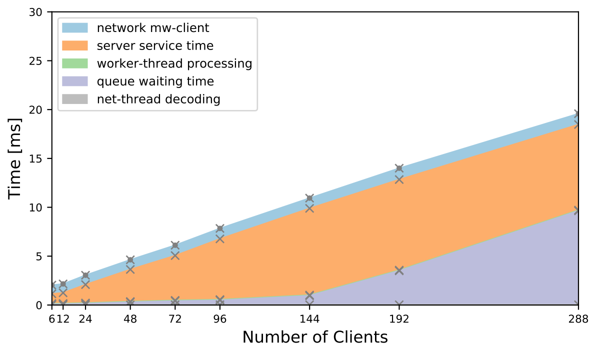


(b) 64 worker-threads

Figure 15: Utilization $\frac{\text{busy time}}{\text{total time}}$ as a function of number of clients in a write-only workload with 2 MWs.



(a) 16 worker-threads



(b) 64 worker-threads

Figure 16: Response time components as a function of number of clients for SET requests with 2 MWs.

3.3 Summary

Maximum throughput for one middleware.

	Throughput [ops/sec]	Response time [ms]	Average time in queue [ms]	Miss rate
Reads: Measured on middleware	2814	1.2	0.1	0.0
Reads: Measured on clients	2780	2.2	n/a	0.0
Writes: Measured on middleware	12030	9.8	4.6	n/a
Writes: Measured on clients	12144	11.8	n/a	n/a

Maximum throughput for two middlewares.

	Throughput [ops/sec]	Response time [ms]	Average time in queue [ms]	Miss rate
Reads: Measured on middleware	2869	1.2	0.1	0.0
Reads: Measured on clients	2827	2.1	n/a	0.0
Writes: Measured on middleware	13569	12.8	3.5	n/a
Writes: Measured on clients	13776	14.0	n/a	n/a

Analysis As shown in sections 3.1 and 3.2 the read-only workload is independent of the number of middlewares in the setup with one server VM and thus in the follwing results are summarized for both systems together. The optimal number of clients for the read-only workload is somewhere in between 6 and 12 clients but the configuration with 12 clients is already strongly affected by the bandwidth bottleneck and results in a response time that is almost two times as long with only a small gain in throughput. So under the evaluated configurations the maximum throughput is achieved with 6 clients and is independent of the number of worker-threads. Thus there is no point in using more than 8 workers. There are no misses because before every experiment *memcached* is initialized with all keys and thus misses have no influence on the response time. Since in the optimal configuration the number of worker threads is greater than the number of clients, the queue basically remains empty and consequently the average time of a request in the queue is a negligible factor in the response time.

For a write-only workload the maximum throughput in the system with a single middleware is achieved with 144 clients and 64 middleware worker-threads. In the system with two middlewares the point of throughput saturation is reached at 192 clients when using 64 worker-threads per middleware. The maximum throughput of the system involving two middlewares comes close to the maximum throughput of 14000 ops/sec recorded in the baseline without a middleware[2] It can be concluded that the overhead of using a middleware in a write-only workload with a single server is minimal, because the baseline without MW poses an upper bound on what can be achieved for a write-only workload.

Since in both setups there are more clients than worker-threads, requests usually spend some time waiting in the queue. However, the additionally gained throughput outweighs the longer response time incurred by this waiting time. The difference between response time measurements on the client and the middleware is between 1 and 2 milliseconds for both workloads which is consistent with the round trip time measurements between the involved VMs. The small difference in throughput measurements can be explained by the exclusion of warm up and cooldown phase in the middleware.

One middleware vs. two middlewares For the read-only workload having one or two middlewares does not make a difference because the server VM network bottleneck is not affected by the number of middlewares in the system.

For a write-only workload the maximal throughput achieved with two middlewares is higher than with a single middleware but this is only an indirect consequence of having an additional middleware because the maximum total number of workers evaluated in the two systems is different and this is the important parameter for the performance as outlined in section 3.2. The maximal throughput in the system with two middlewares is achieved with 64 worker-threads per middleware and thus 128 workers in total. The configuration with a single middleware and 128 worker-threads was not evaluated but the data indicates that this could produce a similar performance assuming the middleware VM does not start to get a problem with context switches or another effect slowing down individual threads.

Key take-away messages

- the number of worker-threads is the main factor for performance in a write-only workload
- with two middlewares and the resulting 128 worker-threads, the maximal throughput of the baseline without middleware is almost reached
- round trip time between client and middleware VM is varying between 1 and 2 milliseconds

[2]The experiments of this section were run after restarting the VMs used in the baseline without middleware and so the measurements are only approximately comparable.

4 Throughput for Writes (90 pts)

4.1 Full System

In this set of experiments, three load generating VMs are connected to a system consisting of 2 MWs and 3 server VMs. The number of clients is varied between 6 and 384 and the performance of a write-only workload is evaluated for configurations with 8, 16, 32 and 64 worker-threads per MW. The details of the configuration are shown in the table below. The interactive law was checked and holds for all experiments in this section (Tab. A.0.3)

3 client VMs, 2 middleware VMs and 3 server VMs

Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	[1, 2, 4, 8, 12, 16, 24, 32, 48, 64]
Workload	Write-only
Worker threads per middleware	[8, 16, 32, 64]

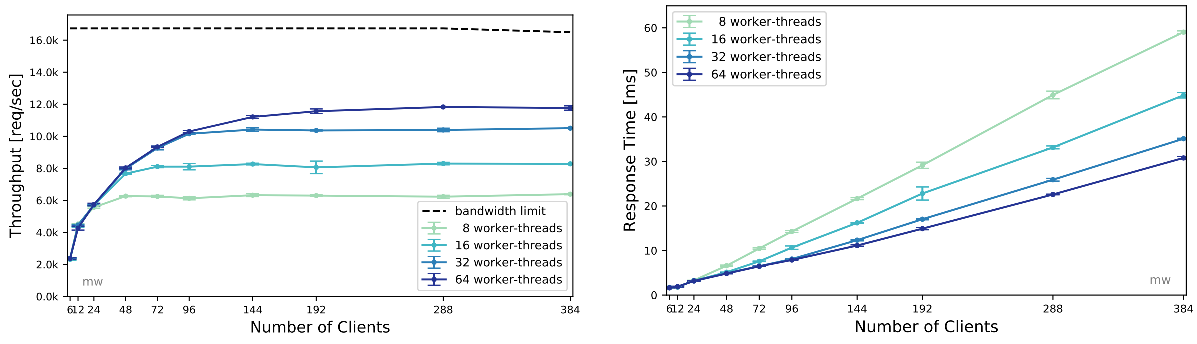
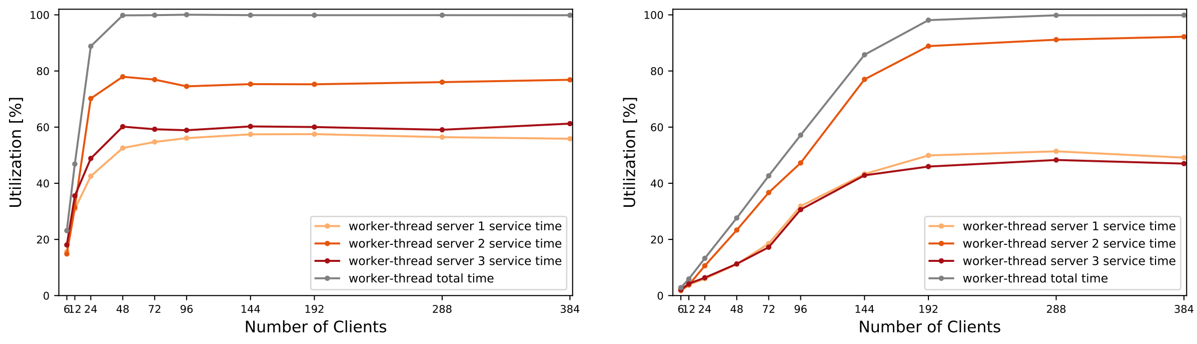


Figure 17: Throughput and response time in a write-only workload in the full system with different number of worker-threads per MW as a function of the number of clients. Both graphs show the sample standard deviation of the measurements over the repetitions as an error metric.

4.1.1 Explanation

As shown in figure 17 the number of worker-threads heavily influences the throughput of the full system. This is consistent with the results from the MW baseline in section 3.2. However, the maximal throughput of the full system with three servers is considerably smaller than the maximal throughput from the baseline using only a single server VM. This is because a SET request relies on the response from all 3 servers and so the server taking the longest time determines the server service time of the request. The effect is amplified because despite the fact that all server VMs are running with the exact same configuration, the service time of server 2 is considerably longer across all experiments than the service time of server 1 and 3 (Fig. 19). The difference is due to the placement of the VMs in the cloud. In consequence the faster service times of server 1 and 3 are almost irrelevant because for most SET requests the performance of server 2 determines the service time.

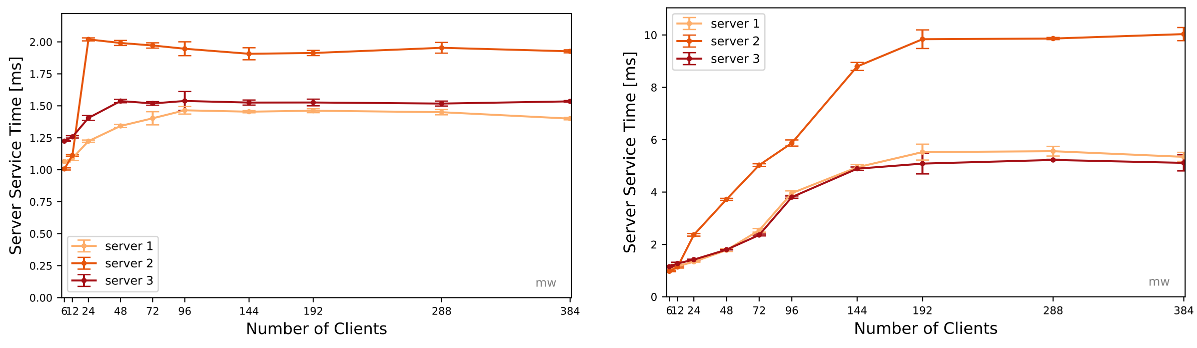
The bottleneck of the system is the number of worker-threads waiting for the slowest server to respond. This becomes evident when considering the utilization of the different components of the system in figure 18. This also explains why the throughput for different number of workers saturates for a different number of clients. The utilization of the worker-thread total time is higher than the utilization of each individual server because even though for most requests server 2 was the slowest, in some cases also one of the other servers was slower and hence the total worker-thread time is on average larger than the server 2 service time. The server service time is increasing in the number of clients up to the point where all worker-threads are almost constantly busy and remains more or less constant from there (Fig. 19) because the number of worker-threads puts a limit on the maximal load a server VM incurs. At the point where all worker-threads are constantly busy, incoming decoded requests need to wait in the queue. These two effects combined lead to an increasing response time in the number of clients as shown in figure 17. This mechanism was already analysed in detail in the MW baseline section 3.



(a) 8 worker-threads

(b) 64 worker-threads

Figure 18: Utilization $\frac{\text{busy time}}{\text{total time}}$ for a write-only workload in the full system



(a) 8 worker-threads

(b) 64 worker-threads

Figure 19: Average server service time per server as a function of number of clients in the full system for a write-only workload. The error metric is the sample standard deviation over the repetitions.
.

4.2 Summary

Maximum throughput for the full system

	WT=8	WT=16	WT=32	WT=64
Throughput [ops/sec] (Middleware)	6262	8102	10154	11206
Throughput [ops/sec] (Derived from MW response time)	6057	8081	10186	11562
Throughput [ops/sec] (Client)	6389	8250	10242	11391
Average time in queue [ms]	4.0	3.6	2.1	1.3
Average length of queue	12	15	11	8
Average time waiting for memcached [ms]	2.4	3.7	5.7	9.6

Throughput The maximum throughput with 8 worker-threads is reached with 48 clients, for 16 worker-threads with 72 clients, for 32 worker-threads with 96 clients and for 64 worker-threads with 144 clients. The system is under-saturated when there are less clients than in the maximum throughput configuration and for more clients the system is saturated. The system is relatively stable because even with only 8 worker-threads and 384 clients the over-saturated phase with decreasing throughput is not reached. The response time shows the familiar linear increase due to the extended server service time/queue waiting time.

As described in section 2 the interactive law[3] $X = \frac{N}{R+Z}$ cannot be applied directly when using the middleware response time because it does not include the time to transfer the request and the response between client VM and middleware VM. To overcome this issue the estimate of the round trip time measured before the set of experiments is used as client thinking time Z. This is a sensible choice because in the viewpoint of the middleware the next request arrives not before the response of the previous request arrived at the client VM and the new request was transferred to the middleware VM and so it appears as if the client is waiting for this time. The deviations between measured throughput in the middleware and derived throughput from middleware response time is explained by the fact that the round trip time is only an estimate and the cloud is not an isolated environment and so the network performance varies over time. In particular for smaller number of clients and shorter response times, errors in the estimate lead to slightly bigger differences. The deviations between middleware and client throughput measurements are consistent with the differences observed in other experiments, but overall the differences are not significant.

Number of Worker-Threads The collected data clearly shows that a larger number of worker-threads lead to a higher throughput because a worker-thread is blocked waiting for all server responses to return and so adding additional worker-threads leads to more concurrency in the system. Consequently for the same number of clients, requests spend less time in the queue which outweights the negative effect that the average time waiting for *memcached* increases in the number of worker-threads.

In the configuration with maximal throughput the queue is always small because otherwise the average time spent in the queue would lead to a significantly longer response time without much benefit in terms of throughput because already in the current state worker-threads are working almost all the time.

Key take-away messages

- for each SET request, the slowest server determines the server service time. Hence the performance of the system with a single server cannot be reached.
- placement of server VM important factor for performance
- number of worker-threads is important factor for performance

[3]In a closed system with N clients the throughput X can be derived using the response time R and the client thinking time Z

5 Gets and Multi-gets (90 pts)

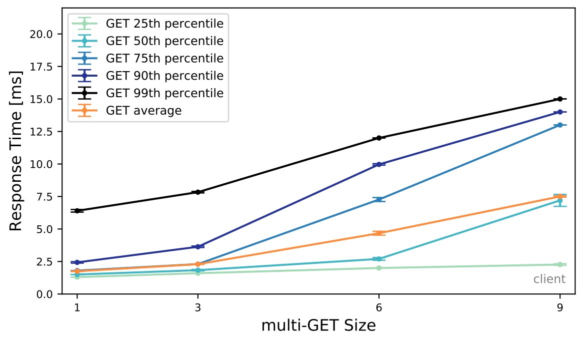
In this set of experiments the behaviour of sharded and non-sharded mode for different number of keys in GET requests are evaluated using 3 load generating machines, 2 MWs and 3 *memcached* servers. Each *memtier* client instance uses 2 virtual clients and so there are 6 clients per MW and hence 12 clients in total.

Results from the baseline with 2 middlewares in section 3.1 show that for 12 clients there is no difference in throughput between 8, 16, 32 and 64 worker-threads and so the configuration with 8 worker-threads is used. This result is also to be expected when considering that in a closed system there can never be more requests than clients at the same time and so with 8 worker-threads there are more worker-threads than clients per MW and so for every decoded request there will always be an idle worker available for processing and the queue is always empty. In consequence, having more worker-threads cannot improve the throughput independent of the number of keys in the request or sharded / non-sharded mode. Details of the configuration are listed in the table below. For all experiments the average number of keys per multi-GET request was recorded in order to ensure that the results between different number of keys can be compared. The recordings show that in the used *memtier* version, each request contains exactly the specified number of keys.

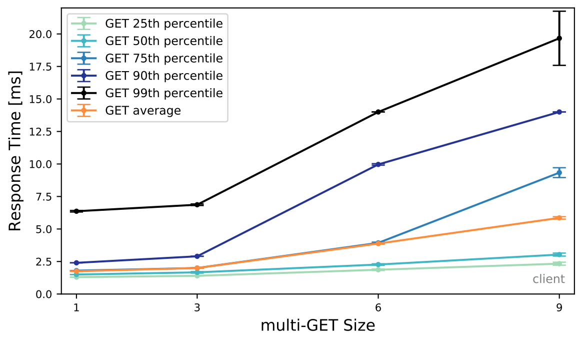
3 client VMs, 2 middleware VMs and 3 server VMs

Instances of memtier per machine	2
Threads per memtier instance	1
Virtual clients per thread	2
Workload	ratio=1:<Multi-Get size>
Multi-Get behavior	Sharded and Non-Sharded
Multi-Get size	[1, 3, 6, 9]
Worker threads per middleware	8

In this setting the focus lies on GET and multi-GET requests because the performance of SET requests in the same setting was already extensively analysed in section 4. However, in order to apply the interactive law to validate the throughput and response time measurements for the read workload, the average SET response time as measured on the client is used as client thinking time. This reflects the fact that *memtier* alternately generates a GET/multi-GET and a SET request. So in the viewpoint of the middleware, the client waits the round trip time and the SET response time before sending the next GET/multi-GET request. As table A.0.4 shows, the interactive law holds for all measurements in this section.

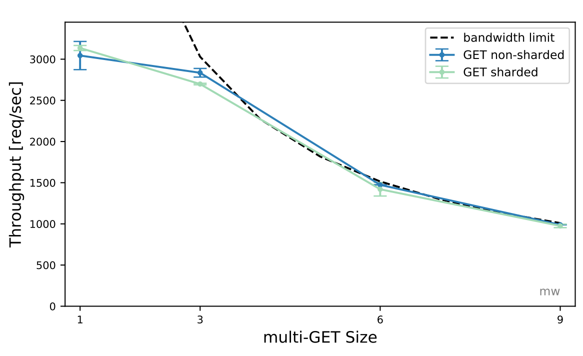


(a) sharded

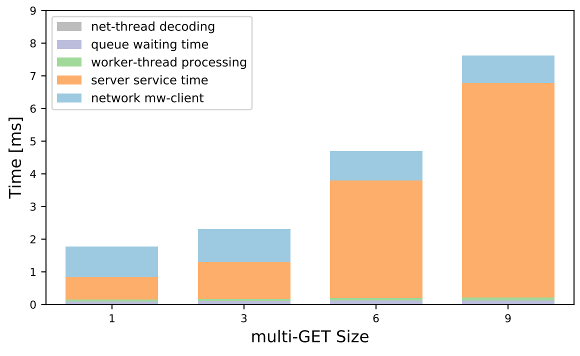


(b) non-sharded

Figure 20: Average response time and 25th, 50th, 75th, 90th and 99th percentiles of different multi-GET sizes. The error metric is the sample standard deviation over repetitions.

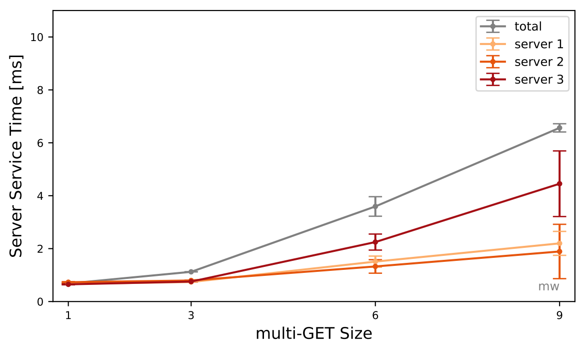


(a) Throughput of multi-GET requests

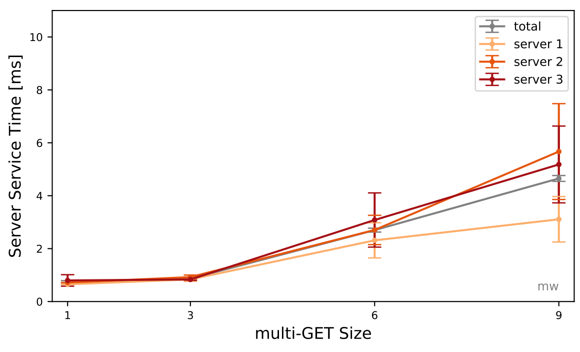


(b) Sharded multi-GET response time decomposition

Figure 21: Throughput of multi-GET requests with bandwidth limit and sample standard deviation over the experiment repetitons as the error metric. On the right side the response time decomposition of sharded multi-GET requests is presented.



(a) sharded



(b) non-sharded

Figure 22: Server service time for each server individually and the average total server service time. All measurements are accompanied with the sample standard deviation over the experiment repetitons.

5.1 Sharded Case

This section evaluates the sharded mode for multi-GETs with 1, 3, 6 and 9 keys.

5.1.1 Explanation

Figure 21a shows that multi-GETs with more than 3 keys are limited by the network upload bandwidth of the server VMs. The 3 servers have a combined upload bandwidth of 297.8 MBit/sec. Since every 4096B value has to be transported from a server VM to a middleware VM independent of sharded or non-sharded mode, a request with 6 keys requires the transportation of at least 24576B without considering any additional overhead. This limits the throughput to 1514 ops/sec which is reached in both the sharded and non-sharded case and thus heavily influences the results.

The network bottleneck becomes also evident when considering the percentiles and the average of the response time in figure 20a. The 75th, 90th and 99th percentile and the average response time have a clear knee between multi-GET requests with 3 and 6 keys. Then for multi-GETs with 9 keys the knee is also visible in the 50th percentile. Only the 25th percentile remains stable, which means that at least 25% of the multi-GET requests are basically not affected by the bottleneck for multi-GET requests with 9 or less values.

In sharded mode it could be expected that there is a trade-off between additional worker-thread processing time disassembling and assembling the multi-GET request and a shorter server service time due to the smaller number of values that need to be handled by a single *memcached* server. However as figure 21b shows the worker-thread processing time is a negligible factor in sharded mode. It is the server service time that dominates the response time. Here a problem of the sharded mode arises, a worker-thread needs to wait for the responses of all involved *memcached* servers and thus the slowest server determines the total server service time of the request. The effect is clearly visible in figure 22a where server 3 is considerably slower in particular for requests with more keys and hence the total server service time is also long. The average total server service time is even worse than the average server service time of server 3 because for some requests server 3 is faster than server 1 or 2 but then the limiting factor is server 1 or 2. Effectively the slowest responding server is the bottleneck of the system and the problem is aggravated by the network bottleneck for multi-GETs with more than 3 values. The same phenomena was previously observed in section 4 for a write-only workload with 3 servers.[4]

5.2 Non-sharded Case

This section evaluates the non-sharded mode for multi-GETs with 1, 3, 6 and 9 keys.

5.2.1 Explanation

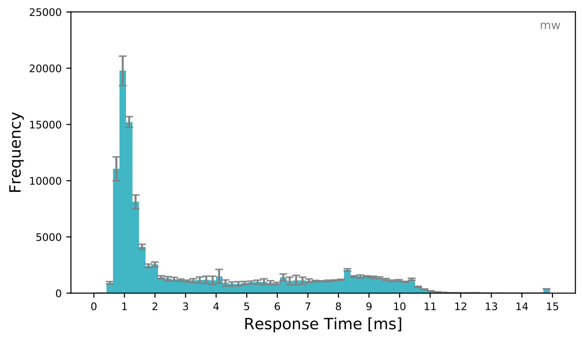
The network bottleneck identified in the sharded mode also directly applies to the non-sharded mode and so the analysis of results for multi-GETs with more than 3 keys has to take this into consideration.

The network bottleneck directly influences the response time as seen in figure 20b, because for the average, the 90th and 99th percentile there is a knee between 3 and 6 keys per multi-GET request. The 75th percentile has the knee at 6 keys per request and the 25th and 50th percentile remain stable up to 9 keys. This shows that at least 50% requests are almost not affected by the network bottleneck which is more than in sharded mode. The fact that less requests are affected in non-sharded mode by the bottleneck can be explained by a simple probability model. Under the simplifying assumption that each message from the server is delayed with probability of 0.5. This is a reasonable assumption because the 50th percentile of the non-sharded mode shows that at least half of the requests are not delayed. The probability of a non-sharded request not being delayed is 0.5, because it relies only on a single message from a server. In sharded mode each request relies on 3 messages from a server and so the probability of at least one of those being delayed is $(1 - 0.5^3) = 0.875$ which results in the whole request being delayed.

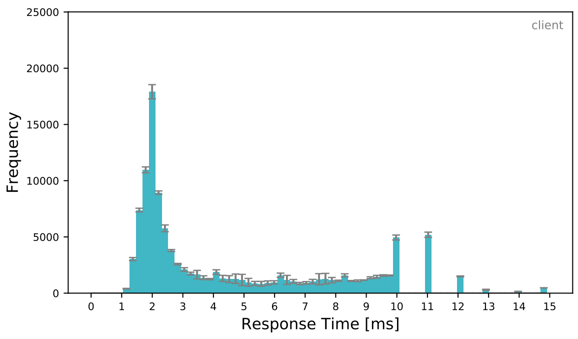
Despite the fact that in non-sharded mode less requests are delayed, figure 20b also shows that if a request is affected by the bottleneck in non-sharded mode, the effect on response time can be much more drastic. This can be seen by the percentile rank of the 99th percentile for multi-GETs with 6 and 9 keys. Additional support for this claim comes from the less stable server service time (Fig. 22b).

Further it can be observed that the average total server service time in non-sharded mode is less affected by the slowest server because the round-robin scheme takes care of the fact that only every third request is sent to this server. This shows the potential of a more sophisticated load balancing scheme considering the current throughput obtained by each server VM.

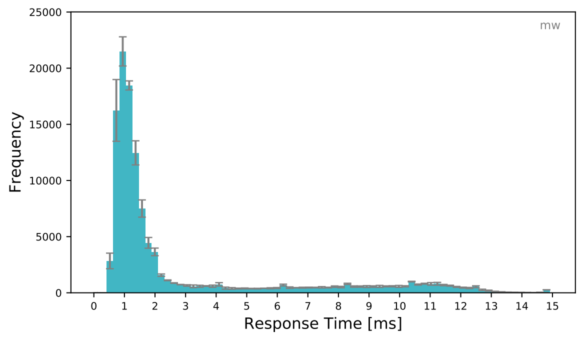
[4]Since the VMs were restarted for experiments in this section, another server is the slowest but the same phenomena occurred.



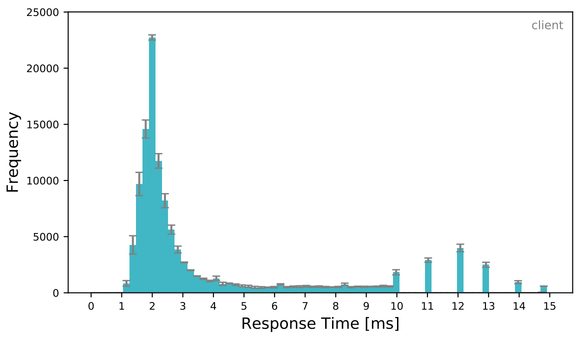
(a) Sharded - Middleware



(b) Sharded - Client



(c) Non-Sharded - Middleware



(d) Non-Sharded - Client

Figure 23: Sharded and non-sharded response time distribution for multi-GETs with 6 keys as measured inside the MW and on the client. Each bucket is showing an error bar indicating the standard deviation in the bucket size over the repetitions.

5.3 Histogram

The constant bucket size of 0.21 was selected by applying the *Freedman–Diaconis rule* (bucket size $= 2\frac{IQR(x)}{\sqrt[3]{n}}$ where $IQR(x)$ is the interquartile range) on the MW response time measurements of the sharded case. All outliers with more than 15 ms are collected in the last bucket.

The histogram contains also the response time measurements from the warmup and cooldown phase to allow a fair comparison between client and MW because *memtier* does not allow to filter them out. However, a manual check of average response time over time indicated that the warmup and cooldown phase only have a limited influence on response time.

As previously explained the response time measured on the client is larger than on the middleware by approximately 1 millisecond resulting in a shift of the time distribution in the histogram. For response times greater than 10ms the *memtier* clients reduced the granularity of the buckets resulting in the artefacts visible in the tail of the client measurements.

Consistent with the results from the previous section the data in figure 23 indicates that the non-sharded case has a longer tail in the response time distribution than the sharded case but at the same time generally also more requests have a small response time than in the sharded case.

5.4 Summary

Number of Keys For values of size 4096B the expected gain in throughput in keys per second is nullified because of the network bottleneck. Hence using multiple keys per GET request does not result in a higher performance in presence of the network bottleneck.

Sharded vs. Non-Sharded Due to the network bottleneck a fair comparison between sharded and non-sharded is difficult. However with 3 keys per multi-GET request the collected data suggests that using the non-sharded mode is better because the gain in server service time as a consequence of only having one instead of three values to fetch is outweighed by the waiting time for the answer of the slowest server. For 6 and 9 keys per multi-GET request there is a trade-off in presence of the network bottleneck. On average the response time in non-sharded mode is smaller than in sharded mode because as explained in the previous section less requests are affected by the network bottleneck. However using the non-sharded mode the tail in the response time distribution is longer and so the 99th percentile rank is significantly higher than in the sharded mode. This results in large response times for some requests. So for multi-GETs with 6 and 9 keys, the preferred option in presence of the network bottleneck depends on whether the application requires to be good on average or having a shorter tail.

Client vs. Middleware Measurements The histograms in figure 23 show that the response time measurements follow the same distribution on client as on the middleware however with a shift of approximately 1 milliseconds representing the transfer between client VM and middleware VM and vice versa. This corresponds with the estimate of the round trip time measured before the experiments.

Key take-away messages

- average response time performance better in non-sharded mode.
- sharded mode for multi-GETs can be used to reduce the 99th percentile of the response time in presence of a server VM network bottleneck.
- in sharded mode the performance of the slowest server is crucial for the response time. In non-sharded mode this effect is much less severe.

6 2K Analysis (90 pts)

A $2^k r$ factorial design analyses the effect of k factors each with 2 levels in an experiment with r repetitions. In this section the primary factors are number of *memcached* servers, number of middlewares and number of worker-threads per middleware. They are investigated using a $2^3 3$ factorial experimental design for both a read-only and write-only workload. The table below lists the two levels of each factor. The additive model in equation 2 is used to model the response variable because the effect between number of servers and middleware VMs / worker-threads is additive. However, the relation between number of middlewares and number of worker-threads can also be seen as multiplicative but to keep the model simple the additive model was chosen such that the majority of relations are correctly modelled.

Factor S: servers	Factor M: middlewares	Factor W: worker-threads
$x_S = \begin{cases} -1 & \text{for 1 server} \\ 1 & \text{for 3 servers} \end{cases}$	$x_M = \begin{cases} -1 & \text{for 1 MW} \\ 1 & \text{for 2 MWs} \end{cases}$	$x_W = \begin{cases} -1 & \text{for 8 WT} \\ 1 & \text{for 32 WT} \end{cases}$

$$y = q_0 + q_S x_S + q_M x_M + q_W x_W + q_{SM} x_S x_M + q_{SW} x_S x_W + q_{MW} x_M x_W + q_{SMW} x_S x_M x_W + e \quad (2)$$

Secondary factors such as number of clients = 192, number of client VMs = 3, and value size = 4096 remain fixed in order to keep the analysis simple, even though it is known from previous sections that they affect performance. The details of the configuration is shown in the following table.

3 client VMs, 1 and 2 middleware VMs and 1 and 3 server VMs

Instances of memtier per machine	1 (1 middleware) or 2 (2 middlewares)
Threads per memtier instance	2 (1 middleware) or 1 (2 middlewares)
Virtual clients per thread	32
Workload	Write-only and Read-only
Worker threads per middleware	8 and 32

Two different performance metrics were computed using simulation: average throughput in ops/sec and average response time in milliseconds. The interactive law characterizes the relation between them in a closed system and since both number of clients and client thinking time remain fixed, the difference of percentage of variation for a factor should be similar in both metrics. Additionally to checking that the percentage of variations are similar, the interactive law was also verified directly with the obtained measurements and as expected the interactive law holds. (Tab. A.0.4)

Methodology Effects q of the factor combinations were calculated using the sign table method using the average $\hat{y}_i$ over the 3 repetitions. The total variation of the data is defined as:

$$SST = \sum_{i=1}^{n}\sum_{j=1}^{r}(y_i - \bar{y})^2 = \underbrace{nrq_S^2}_{SSS} + \underbrace{nrq_M^2}_{SSM} + \cdots + \underbrace{nrq_{SM}^2}_{SSSM} + \cdots + \underbrace{nrq_{SMW}^2}_{SSSMW} + SSE \qquad (3)$$

with $n = 2^3$, $r = 3$ and mean response from all repetitions of all experiments $\bar{y}$. An estimation of the experimental error is done by calculating $SSE = \sum_i^{2^3}\sum_j^3 e_{ij}^2$ where $e_{ij} = y_{ij} - \hat{y}_i$ measures the difference between the result of the $j'th$ repetition and the average obtained in experiment i. This allows to calculate the percentage of variation $\frac{SSA}{SST}$ which is the fraction of the variation explained by a factor A. Additionally for the effect of each factor the 90% confidence interval using the t-value 1.746 was calculated. The impact of a factor is significant if 0 is not in the confidence interval. Details of the $2^k r$ factorial design can be found in [5].

Write-Only The experimental measurements of throughput and response time are listed in table 1. The percentage of variation of the error is small which shows that the variation between the configurations is explained by the different factors rather than noise.

Table 2 shows that the performance of the system in a write-only workload decreases when using 3 servers. This is a familiar problem known from section 4 because the total server service time for SET requests is determined by the slowest server. The problem is aggravated because server 2, as previous experiments run on the same set of VMs have already shown, is constantly slower than the other servers. This shows in the average service times of each server which are 1.8 ms respectively 2.0 ms for server 1 and server 2 compared to 3.0 ms for server 2.

The number of middlewares is an important factor which explains 24.3% of the variation. Using 2 middlewares the throughput is increased because as argued in section 3 the resulting increase in worker-threads is beneficial to the write-only workload with 192 clients.

The strongest influence on performance is the number of worker-threads. The percentage of variation of factor W (worker-threads) is higher than of factor M (middleware) because of the chosen levels. If only one of the factors could be on the upper level, factor W would be the better choice because then there are 32 workers in the system in total compared to only 16 when there are 2 middlewares but only 8 workers each. This claim is also supported by the collected data in table 1 which shows that as expected ideally both factors are on the upper level.

The interaction between the different factors does not play a major role in the explanation of the measured throughput and response times.

									Throughput (ops/sec)		**Response Time** (ms)	
i	I	S	M	W	SM	SW	MW	SMW	(y_{i1}, y_{i2}, y_{i3})	$\hat{y_i}$	(y_{i1}, y_{i2}, y_{i3})	$\hat{y_i}$
1	1	-1	-1	-1	1	1	1	-1	(6620, 6258, 6285)	6387	(28.0, 29.7, 29.5)	29.1
2	1	1	-1	-1	-1	-1	1	1	(4389, 4455, 4457)	4434	(42.6, 41.6, 42.0)	42.1
3	1	-1	1	-1	-1	1	-1	1	(9328, 9493, 9610)	9477	(19.6, 19.2, 19.0)	19.2
4	1	1	1	-1	1	-1	-1	-1	(6327, 6216, 6388)	6310	(29.3, 29.6, 29.0)	29.3
5	1	-1	-1	1	1	-1	-1	1	(10359, 10445, 10434)	10413	(16.9, 16.8, 17.0)	16.9
6	1	1	-1	1	-1	1	-1	-1	(7579, 7566, 7505)	7550	(23.4, 23.5, 23.2)	23.4
7	1	-1	1	1	-1	-1	1	-1	(12872, 12690, 12735)	12766	(13.7, 14.0, 13.9)	13.8
8	1	1	1	1	1	1	1	1	(10416, 10480, 10265)	10387	(17.0, 16.9, 17.3)	17.1

Table 1: Measurements of the $2^3 3$ experimental design for a write-only workload

	Throughput (ops/sec)				**Response Time** (ms)			
Factor	Effect	Sum of Squares	Percentage of Variation	Confidence Interval 90%	Effect	Sum of Squares	Percentage of Variation	Confidence Interval 90%
I	8465	1719918k		$(8427, 8504)$	23.9	13656		$(23.7, 24.0)$
S	-1295	40262k	25.3	$(-1334, -1256)$	4.1	401	22.1	$(3.9, 4.2)$
M	1269	38677k	24.3	$(1231, 1308)$	-4.0	383	21.1	$(-4.1, -3.8)$
W	1813	78915k	49.5	$(1775, 1852)$	-6.1	884	48.6	$(-6.2, -5.9)$
SM	-91	199k	0.1	$(-130, -52)$	-0.8	14	0.8	$(-0.9, -0.6)$
SW	-15	5k	0.0	$(-54, 24)^a$	-1.7	67	3.7	$(-1.8, -1.5)$
MW	28	19k	0.0	$(-11, 67)^a$	1.7	66	3.6	$(1.5, 1.8)$
SMW	212	1079k	0.7	$(173, 251)$	-0.0	0	0.0	$(-0.2, 0.1)^a$
Error		$189k$	0.1			2.8	0.2	

Table 2: Effect and percentage of variation of factor combinations in a write-only workload. The effect of factors showing an *a* are not significant.

Read-Only Response time and throughput measurements for the different configurations in the read-only workload are listed in table 3. The percentage of variation of the error is small which indicates that the differences between the configurations are not explained by noise.

The number of servers is the dominant factor in a read-only workload because as shown in the middleware baseline in section 3 when using a single server VM, the throughput is bound to around 3000 ops/sec by the upload bandwidth of the server VM. Naturally when using three server VMs the upload bandwidth capacity triples, which leads to a much higher throughput and lower response time. This explains why factor S (server), is responsible for basically all the percentage of variation. (Table 4)

Adding an additional MW or increasing the number of workers has only a marginal positive effect on the performance in presence of the network bottleneck without increasing the number of servers. This is consistent with observations in previous sections. However, the collected data in table 3 indicates that given there are 3 servers and at least 16 worker-threads in total in the system, the throughput reaches the bandwidth limit of approximately 9000 ops/sec of the 3 server VMs. The 16 worker-threads in total are reached by either using 32 threads in a single middleware or then with 2 middlewares with at least 8 worker-threads per MW. Consequently it is beneficial to the performance of the system in a read-only workload to use 32 worker-threads or 2 middlewares if 3 servers are used. The simple additive model and also the multiplicative model which is not presented here, fail to capture this effect.

The combined effects are in general really small and so the interaction between the different factors is small. As expected by the interactive law all factors having a positive effect on throughput have a negative effect on the response time and the percentage of variation is proportional to each other.

									Throughput (ops/sec)		**Response Time** (ms)	
i	I	S	M	W	SM	SW	MW	SMW	(y_{i1}, y_{i2}, y_{i3})	$\hat{y}_i$	(y_{i1}, y_{i2}, y_{i3})	$\hat{y}_i$
1	1	-1	-1	-1	1	1	1	-1	(2937, 2947, 2941)	2942	(64.3, 64.1, 64.2)	64.2
2	1	1	-1	-1	-1	-1	1	1	(6793, 6946, 6966)	6902	(27.3, 26.6, 26.5)	26.8
3	1	-1	1	-1	-1	1	-1	1	(2946, 2952, 2941)	2946	(62.2, 64.0, 64.2)	63.4
4	1	1	1	-1	1	-1	-1	-1	(8805, 8810, 8801)	8805	(20.7, 20.5, 20.7)	20.7
5	1	-1	-1	1	1	-1	-1	1	(2948, 2938, 2944)	2943	(63.8, 63.9, 63.8)	63.8
6	1	1	-1	1	-1	1	-1	-1	(8810, 8819, 8802)	8811	(20.5, 20.4, 20.5)	20.4
7	1	-1	1	1	-1	-1	1	-1	(2955, 2944, 2952)	2950	(63.6, 63.8, 63.6)	63.6
8	1	1	1	1	1	1	1	1	(8798, 8807, 8801)	8802	(20.6, 20.5, 20.0)	20.4

Table 3: Measurements of the $2^3 3$ experimental design for a read-only workload

	Throughput (ops/sec)				**Response Time** (ms)			
Factor	Effect	Sum of Squares	Percentage of Variation	Confidence Interval 90%	Effect	Sum of Squares	Percentage of Variation	Confidence Interval 90%
I	5638	762787k		$(5626, 5650)$	42.9	44204		$(42.8, 43.1)$
S	2692	173961k	95.5	$(2680, 2704)$	-20.9	10439	99.1	$(-21.0, -20.7)$
M	238	1363k	0.7	$(226, 250)$	-0.9	19	0.2	$(-1.0, -0.7)$
W	239	1371k	0.8	$(227, 251)$	-0.8	17	0.2	$(-1.0, -0.7)$
SM	235	1330k	0.7	$(223, 248)$	-0.7	10	0.1	$(-0.8, -0.5)$
SW	237	1353k	0.7	$(225, 250)$	-0.8	15	0.1	$(-1.0, -0.6)$
MW	-239	1367k	0.8	$(-251, -227)$	0.8	16	0.2	$(0.7, 1.0)$
SMW	-239	1374k	0.8	$(-251, -227)$	0.7	11	0.1	$(0.5, 0.8)$
Error		$18k$	0.0			3.0	0.0	

Table 4: Effect and percentage of variation of factor combinations in a read-only workload.

Key take-away messages

- using multiple servers in a write-only workload decreases performance
- for a write-only workload, the previously analysed effect of number of workers-threads on throughput and response time also shows in 2K analysis
- for a read-only workload, the importance of multiple servers because of the network bandwidth problem is also shown by the 2K analysis

7 Queuing Model (90 pts)

The results of this section were obtained by using queueing models for the system configurations analysed in section 3 and 4. It was already shown that the interactive law holds for the measurements from both sections and so an additional check is omitted here. The software [7] was used for the queueing models.

			Number of Clients						
			6	12	24	48	72	96	144
8 WT	$\mu_{M/M/1} = 6385$	$\lambda =$	2357	4520	5562	6262	6244	6173	6320
	$\mu_{M/M/16} = 399$	$\rho =$	0.37	0.71	0.87	0.98	0.98	0.97	0.99
16 WT	$\mu_{M/M/1} = 8313$	$\lambda =$	2367	4335	5699	7664	8103	8201	8266
	$\mu_{M/M/32} = 260$	$\rho =$	0.28	0.52	0.69	0.92	0.97	0.99	0.99
32 WT	$\mu_{M/M/1} = 10504$	$\lambda =$	2397	4456	5708	7975	9267	10155	10420
	$\mu_{M/M/64} = 164$	$\rho =$	0.23	0.42	0.54	0.76	0.88	0.97	0.99
64 WT	$\mu_{M/M/1} = 11848$	$\lambda =$	2405	4312	5738	8027	9339	10300	11210
	$\mu_{M/M/128} = 93$	$\rho =$	0.2	0.36	0.48	0.68	0.79	0.87	0.95

Table 5: Arrival rates λ, service rate μ and traffic intensity ρ for both M/M/1 and M/M/m models for different number of worker-threads

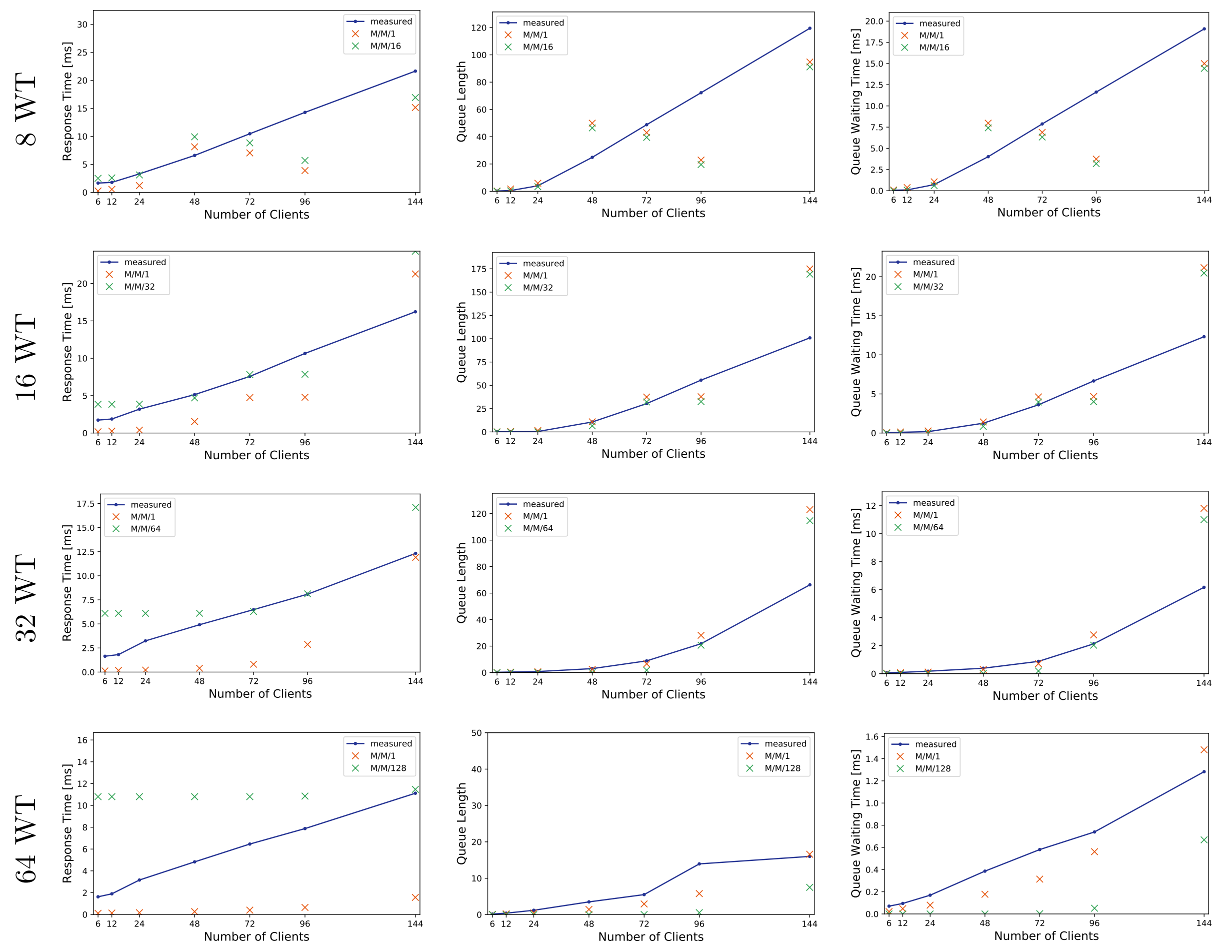


Figure 28: Comparison of response time, queue length and queue waiting time as a function of number of clients for 8, 16, 32 and 64 worker-threads (WT) between measurements, M/M/1 and M/M/m model.

7.1 M/M/1

In this section the entire system from Section 4 (write-only throughput) is modelled using one M/M/1 queue. An M/M/1 queue assumes that the interarrival times and service times of requests are exponentially distributed and that requests are waiting in a possibly unbounded

FCFS queue[5] for processing by the only server in the system.

An M/M/1 queue is completely defined by two parameters, the arrival rate λ and the service rate μ. The average number of requests per second arriving in the system depends on the user load and is measured in the net-thread of each MW. This measurement from both MWs is added up and used as the arrival rate λ for the M/M/1 queue. The service rate μ is independent of the user load and characterizes how fast the system is able to serve requests in a certain configuration. The maximum observed throughput of the system with a certain configuration can be used as an estimate for the service rate because this shows that the system is capable of serving requests at least as fast as this. The service rate μ is defined as the maximum throughput observed for a worker-thread configuration independent of the number of clients.

The chosen parameters are listed in table 5. The system is always stable because the traffic intensity $\rho = \frac{\lambda}{\mu}$ is always smaller than 1. The traffic intensity also represents the utilization of the system and this matches nicely with the results from section 4, where it was shown that the system saturates when there are considerably more clients than total worker-threads in the system[6]. Apart from the traffic intensity the performance of the model is evaluated by comparing it to the measured response time, number of requests in the request queue and waiting time in the queue.

The M/M/1 model constantly underestimates the response time because it assumes there is only a single server (worker-thread) and so the given service rate implies that this single server processes each job rapidly and hence the predicted response time is much lower than the observed response time. In reality the service time of a worker-thread is much larger and the achieved throughput is only possible because multiple worker-threads process requests in parallel. In configurations with relatively few worker-threads the discrepancies in response time are smaller compared to configurations with more working-threads because the concurrency in the system, which the M/M/1 model fails to capture, increases in the number of worker-threads. Both the number of requests in the queue and the waiting time in the queue are approximated well with the M/M/1 model because there the assumption of a FCFS queue matches well with the request queues in the two middlewares. However, this accuracy is only possible because the decoding in the net-thread is not a bottleneck where requests are already queued for a long time before entering the MW request queue. (Fig. 28)

7.2 M/M/m

In this section the entire system from Section 4 (write-only throughput) is modelled using one M/M/m queue for each worker-thread configuration. An M/M/m queue assumes that the interarrival times and service times of requests are exponentially distributed and that requests are waiting in a possibly unbounded FCFS queue for processing by one of the m servers in the system.

An M/M/m queue is completely defined by three parameters, the arrival rate λ, the service rate μ and the number of servers m. The arrival rate λ and the service rate μ are defined as in the M/M/1 model with the only difference that the service rate is divided by the number of servers m in the system because when assuming that there are m servers splitting the workload equally, then each server is responsible for $\frac{1}{m}$ of the total number of serviced requests. The parameter m is chosen such that it matches the total number of worker-threads in the system which is two times the number of worker-threads per middleware since there are two MWs.

[5]first come first served queue

[6]recall that there are 2 MWs in the system and hence the total number of worker-threads is two times the number of worker-threads per MW which are listed in table 5)

It can be expected that the M/M/m queue is a much better model for the response time than the M/M/1 queue because it accounts for the concurrency of the worker-threads. The predictions for the number of requests in the queue and queue waiting time should be relatively similar to the M/M/1 model. The traffic intensity for the M/M/m model is identical to the traffic intensity in the M/M/1 by construction and as seen in the previous section it nicely captures the utilization of the system.

While the system is under-saturated when there are more worker-threads than clients, the M/M/m model is a bad model for the response time. This becomes in particularly evident in figure 28 with 128 worker-threads in total. The reason for this phenomena is that the model assumes a constant service-time. However, the analysis in section 4 showed that in reality the server service time and consequently also the worker service time depends on the number of busy workers which in the under saturated is smaller than in the configuration with maximum throughput.

Despite the fact that in the under-saturated phase the M/M/m model fails to capture the response time accurately, when the system is running at near to optimal load with approximately equal number of worker-threads in total and clients, then the M/M/m model is a good model for the response time and outperforms the M/M/1 model significantly. When there are much more clients per middleware than worker-threads, the accuracy of the model decreases and there can even be some artefacts as observed in figure 28 with 8 worker-threads.

As expected the difference between the predictions for number of requests waiting in queue and waiting time are equally good to the M/M/1 model because the model also accurately models the situation in the 2 middlewares with the FCFS request queue where all requests are waiting for service and no major waiting time happens before that.

7.3 Network of Queues

The results for the network of queues model were obtained using the convolution algorithm for general load-dependent service centers. Details of the algorithm and further references can be found in the documentation of the applied *queueing* software [7].

One Middleware - Model The net-thread is modelled as an M/M/1 queue using the measured decoding time as the service time. This is a sensible choice because as shown in previous sections, the decoding time of the net-thread is independent of the number of clients and all requests have to go through this decoding stage which explains the visit ratio of 1.

The request queue in combination with the m worker-threads is modelled as an M/M/m queue with a visit ratio of 1 because with a single middleware all requests pass through there. Due to the architecture of the middleware, the *memcached* server is not modelled as a separate queue because a worker-thread cannot process a new request until the *memcached* server responded and so the time for a request to the *memcached* server is part of the worker-thread service time. This modelling choice leads to a conflict in the service time of a worker-thread because as shown in previous sections the server service time of a *memcached* server is depending on the number of busy worker-threads and so in an under-saturated system, the worker-thread service time is smaller than in a system where all worker-threads are busy. It was decided to use the measured worker-thread processing time when all worker-threads are busy as the service time for a server in the M/M/m queue. In consequence the model predictions for the throughput are only accurate when there are more clients than worker-threads because as seen in previous sections, the worker-thread processing time is independent of the user-load from the point where there are more users than worker-threads.

The network between client VM and middleware VM and vice versa are modelled as two

Network 1 MW	V	S_{WO}	S_{RO}	Type
Client	1	0.0	0.0	delay center
Network C - MW	1	0.916	0.916	delay center
Net-Thread	1	0.029	0.008	M/M/1
Worker-Threads	1	5.147	2.716	M/M/m
Network MW - C	1	0.916	0.916	delay center

Network 2 MWs	V	S_{WO}	S_{RO}	Type
Client	1	0.0	0.0	delay center
Network C - MW 1	0.5	0.635	0.635	delay center
Net-Thread MW 1	0.5	0.026	0.010	M/M/1
Worker-Threads MW 1	0.5	8.941	5.424	M/M/m
Network MW 1 - C	0.5	0.635	0.635	delay center
Network C - MW 2	0.5	0.635	0.635	delay center
Net-Thread MW 2	0.5	0.026	0.010	M/M/1
Worker-Threads MW 2	0.5	8.941	5.424	M/M/m
Network MW 2 - C	0.5	0.635	0.635	delay center

Figure 29: Network of Queues for 1 and 2 MWs with visit ratio (V) and service time (S) in milliseconds.

separate delay centers each with a service time of half the measured round trip time. In between there is a client delay center with 0 service time which has no influence but fits well with the conceptual model of the system. All three delay centers have a visit ratio of 1 because all requests pass through them. Figure 29 shows the detailed structure of the network of queues with the respective parameters.

One Middleware - Model Performance The network of queues model with a single middleware was applied to both the read-only and write-only workload from section 3.1. In order to evaluate the performance of the model, the predicted and measured throughputs are compared for different number of clients. In addition the utilization of each component as predicted by the model is shown in order to identify the bottleneck component.

For the write-only workload the configuration with 64 worker-threads is used because as shown it leads to the highest throughput. Figure 30a shows that for reasons described in the previous paragraph, the model fails to predict the throughput when there are less clients than worker-threads but afterwards the model is accurate. The component utilizations in figure 30a show that worker-threads become the bottleneck if there are considerably more clients than worker-threads. This is consistent with the results from section 3.1.

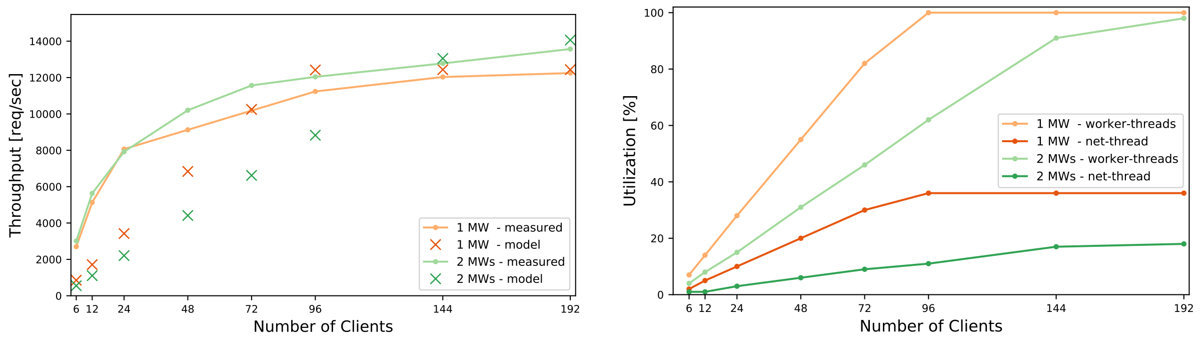
For the read-only workload the configuration with 8 worker-threads is used because as shown there is no point in using more worker-threads because the server VM upload bandwidth limit is already reached with 8 worker-threads. The throughput prediction is accurate as soon as the user load is such that all 8 worker-threads are kept constantly busy (Fig. 30b). This is due to the choice of worker-thread service time as described above . The component utilizations in figure 30b show that the worker-threads are the bottleneck in the model. This is only partially true because as seen in section 3.1 the real bottleneck is the upload bandwidth capacity of the server VM. However, since the limited upload bandwidth capacity between server VM and middleware VM is not explicitly modelled, the model also does not have the power to detect this and instead the limit manifests itself in the worker-thread processing time.

Two Middlewares - Model The network of queues model for the system with two middlewares in section 3.2 is similar to the network of queues with 1 MW. The difference is that all components apart from the client delay center are duplicated and in consequence each of them has a visit ratio of 0.5. This is a reasonable choice because half of the clients use middleware 1 and the other half uses middleware 2. As shown in section 3.2 the *memcached* service time remains constant when the user load is such that all worker-threads in the system are kept constantly busy and so the dependence between the two middlewares can be neglected. The motivation for the choice of the component service times from the model involving one middleware also directly apply to this model with two middlewares. The details of the model are listed in figure 29.

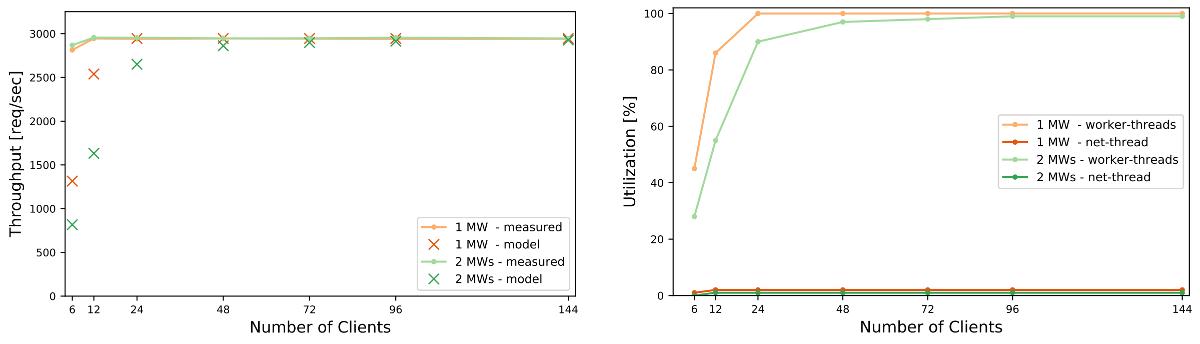
Two Middlewares - Model Performance The network of queues model with a 2 MWs was applied to both the read-only and write-only workload from section 3.2. As in the model with a single middleware, the predicted and measured throughput are compared for different number of clients. And the utilization of each component as predicted by the model is listed in order to identify the bottleneck component. For the write-only workload the configuration with 64 worker-threads per middleware is used and for the read-only workload the configuration with 8 worker-threads per middleware is used because they lead to the maximal throughput.

In the write-only workload, the model is able to predict the throughput accurately when all 128 worker-threads are busy which happens only for more than 128 clients. This is due to the familiar issue with the worker-thread service time that up to then would depend on the user load in the real system but is kept fixed in the model. The component utilizations in figure 30a show that the bottleneck is once again the worker-threads waiting for a server response, which matches the analysis of section 3.2. Compared to the system with 1 MW, the point of saturation is reached for a larger number of clients which manifests itself both in the component utilizations and in the throughput.

In the read-only workload the situation is the same as in the model with one middleware. The throughput predictions match when the number of clients is such that all 16 workers are constantly busy. The bottleneck is identified as the worker-threads which is correct in the sense that the bandwidth bottleneck of the server VM, which was identified as the actual bottleneck in section 3.2, is part of this component. However, in the read-only workload the granularity of the network model does not allow to detect the bandwidth limit of the server VM as the real problem.



(a) write-only



(b) read-only

Figure 30: Comparison of throughput and utilization as a function of number of clients between the two network of queues models and the measurements from the MW baseline.

A Interactive Law

The minor deviations from the interactive law according to equation 1 are a result from small differences in the measured throughput on client and MW. This can be explained because the warmup and cooldown phase are excluded in the MW measurements. For larger response times, the deviations also increase slightly but they remain marginal when putting them in relation to the response time. This confirms that the interactive law holds for all experiments.

A.0.1 Baseline with Middleware: One Middleware

	clients	6	12	24	48	72	96	144	192	288	6	12	24	48	72	96	144
worker	8	0.0	0.0	0.0	0.1	0.3	0.6	-	-	-	0.0	0.1	0.3	0.4	1.3	2.0	1.8
	16	-0.1	0.0	0.0	0.1	0.1	0.2	0.2	0.4	-	0.0	0.0	0.1	0.9	1.2	1.0	1.1
	32	0.0	-0.1	0.0	0.0	0.0	0.1	0.2	0.2	-	0.0	0.0	0.2	0.6	0.5	0.7	1.1
	64	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.9	0.0	0.0	0.1	0.5	0.5	0.7	1.1
		in milliseconds									in milliseconds						
		write-only									read-only						

A.0.2 Baseline with Middleware: Two Middlewares

	clients	6	12	24	48	72	96	144	192	288	384	6	12	24	48	72	96	144
worker	8	0.0	0.0	0.0	0.1	0.2	0.4	0.6	0.7	1.1	-	0.0	0.0	0.2	0.8	1.0	1.4	2.0
	16	0.0	0.0	0.0	0.0	0.1	0.1	0.3	0.5	0.9	-	0.0	0.0	0.3	0.6	1.1	1.6	1.9
	32	0.0	0.0	0.0	0.0	0.1	0.2	0.3	0.4	0.6	-	0.0	0.1	0.4	1.0	1.2	1.6	2.9
	64	-0.1	0.0	0.0	0.1	0.1	0.1	0.3	0.2	0.4	0.3	0.0	0.1	0.3	0.8	1.1	1.4	2.5
		in milliseconds										in milliseconds						
		write-only										read-only						

A.0.3 Throughput for Writes

	clients	6	12	24	48	72	96	144	192	288	384
worker	8	-0.1	0.0	0.0	0.1	0.1	0.3	0.7	1.0	1.6	1.8
	16	-0.1	-0.1	0.0	0.0	0.1	0.0	0.4	0.2	1.0	2.7
	32	-0.1	-0.1	0.0	0.0	0.0	0.1	0.2	0.4	0.7	0.5
	64	-0.1	-0.1	0.0	0.0	0.1	0.1	0.2	0.3	0.3	0.9
		in milliseconds									

A.0.4 Gets and Multi-gets / 2K Analysis

multi-GET size	1	3	6	9
sharded	-0.1	-0.1	0.1	0.4
non-sharded	-0.1	-0.1	0.1	0.3
	in milliseconds			

2^k i	1	2	3	4	5	6	7	8
write-only	0.7	1.2	0.5	1.3	0.5	0.8	0.3	0.3
read-only	3.5	0.4	2.9	0.9	2.0	1.0	2.4	0.9
	in milliseconds							

References

[1] Log4j2 asynchronous loggers. `https://logging.apache.org/log4j/2.x/manual/async.html`. Accessed: 2018-10-12.

[2] Memcached. `https://memcached.org/`. Accessed: 2018-10-12.

[3] Memcached protocol. `https://github.com/memcached/memcached/blob/master/doc/protocol.txt`. Accessed: 2018-10-12.

[4] Memtier benchmark. `https://github.com/RedisLabs/memtier_benchmark`. Accessed: 2018-10-12.

[5] R. Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[6] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[7] M. Marzolla. The qnetworks toolbox: A software package for queueing networks analysis. In K. Al-Begain, D. Fiems, and W. J. Knottenbelt, editors, *Analytical and Stochastic Modeling Techniques and Applications, 17th International Conference, ASMTA 2010, Cardiff, UK, June 14-16, 2010. Proceedings*, volume 6148 of *Lecture Notes in Computer Science*, pages 102–116. Springer, 2010.